Pocket PC Application Builder

User Manual

Contents

Preamble

The writing of applications for devices like PDA or specialized industrial terminals such as barcode terminals is a complex task involving various development and system skills and requiring a lot of software components.
With the Application Builder you will be able to develop far more easily for PDA and specialized industrial terminals under Windows CE environment that will meet the needs of enterprise-scale applications.

An Overview of the Application Builder

The Application Builder is designed for the fast creation of applications that must run on Windows CE® platforms. Thanks to the Application Builder you'll have the option to create applications aimed at different domains. It allows you to develop graphical user interfaces and to set the logic of the application with the help of dialog boxes instead of development language code. This tool makes program development accessible to people with no specific skills. The creation of applications is also done on Windows PC - Test and debugging.. Then, the applications are downloaded to run on the terminal. With this software you don't have to master the design and management of databases. With the Application Builder, we first aimed for ease of data operations declaration. While creating the software, it provides you with the simple mechanisms of performing database processing operations. Applications process the data and stock them with the help of Microsoft® CE Data Access (ADOCE) technology that is the standard for Windows CE® platform. Using this technology, the application will be able to manage Win CE Database. The Application Builder allows you to import and export Microsoft® Access database tables from/to a computer.
Another advantage of the Application Builder: transactions are designed in dialog boxes and are generated under the control of the Application Builder. These transactions can be requests to remote databases, file transfer requests or commands to trigger other remote logical modules running on Host. To do so a Communication Server is used and has to be installed on a Host used as communication Server. The Communication Server is an additional product.
Other available functions with the Application Builder are functions to deal with GSM Data or GPRS communication, FTP,….on the same easy mode.

Installation.

To install the Application Builder you have to:

1. Insert installation CD-ROM
2. Select the product name in the list of products available in the CD user interface
3. Click Install button, and this runs the product setup program. Follow instruction given by the setup application.
4. Once the installation is finished, connect the hard lock key on the printer port LPT1

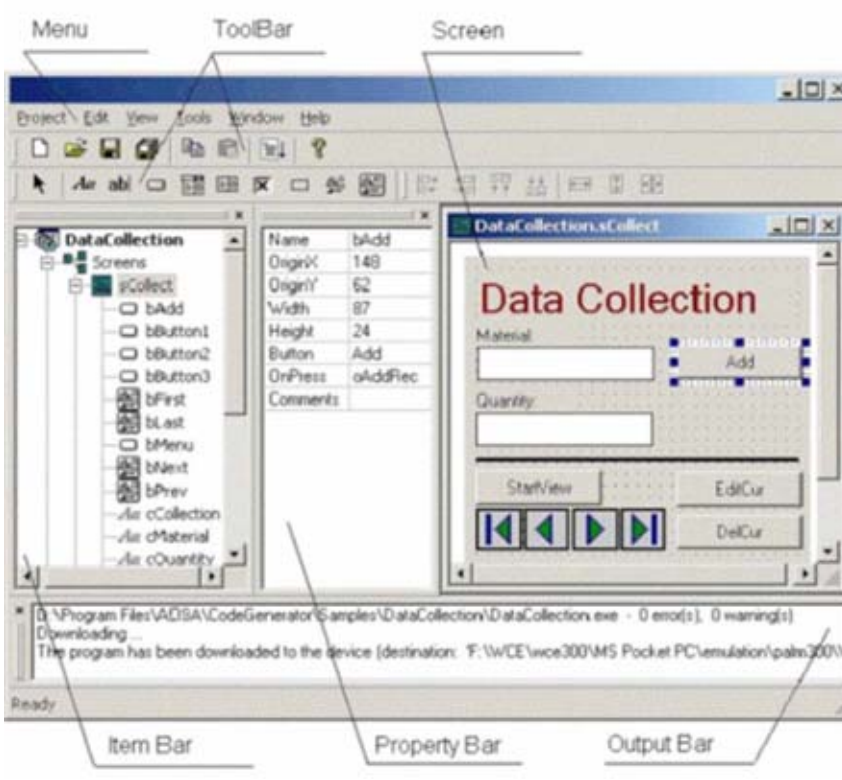
If you have no PDA on which to test the developed applications you can use the Microsoft WindowsCE emulator. This product is available on the CD by installing the *Pocket PC SDK* product.
Installation of the program *Desktop Pocket PC Emulation*:

1. In the *Program Maintenance* dialog box, choose *Custom* option,
2. In the *Custom Setup* dialog box, select only *Common*, and unselect the other ones,
3. Press the button *Finish.*
4. Run the menu Start\Programs\Microsoft Windows® Platform SDK for Pocket PC\Emulation Environment for Pocket PC. In a DOS window is given the directory. Copy into this directory the files that are in the following address:
    a. Disk2:\POCKTSDK\PROGRAM FILES\MFC\LIB\X86EM\mfcce300.dll
    b. Disk2:\POCKTSDK\PROGRAM FILES\MFC\LIB\X86EM\olece300.dll
5. Run *Desktop Pocket PC Emulation*.

If you want to load the applications on the PDA the *ActiveSync* program must be installed. This product is available on the CD by installing *Active Sync* product.

How the Application Builder works

The main window will be displayed when it is started (pict.1). Here, you can find the main components that will become active when you open a new project or an existing one. The first action is to create a new Project. The Project corresponds to a framework of an application to which elementary functions are appended.
The main workspace is made up of several tool bars, each of them having its own function. **Tool Bar** provides access to the most frequently used commands. **Item Bar** shows the list of the open projects as well as of the structure of the objects connected to them. **Property Bar** shows the properties of the selected element, and **Output Bar** reports the results of the latest compilation. The rest of the available space shows the screens of the Project. You set the scale of this area by changing the dimensions of the bars to assign the space you need to each window .



Pict . 1 : main windows of Application Builder

Tools

All the visible elements you can see in the main window are classified into the following groups according to their functions: **Tool Bar** provides fast and easy access to standard commands, **Palet Bar** groups the different controls that can be used in a project; **Dimension Bar** is designed to automatically align the different objects.

Toolbar

	New	Create a new project
	Open	Open an existing Project
	Save	Save the current Project
	Save all	Save all open Projects
	Copy	Copy the selection
	Paste	Paste the selection

The **Copy** function concerns all the links and properties of a selected object. This function is useful when repetitive functions have to be implemented.

Generate button allows you to generate the executable program corresponding to the active
Project.
The *Options* dialog box accessible from the Tools menu allows to indicate that you want to automatically download the Project after generation (check box : *Automatically download the program after its generation*). You can make the grid active or not and can keep controls aligned on the grid.

About button provides information on the product version you are working with.

Picture 2: **Options dialog box**

Pallet bar

Pallet Bar is made up of different visual objects that can be included in your project.
Note: *The visual elements of the project will be referred to as controls from now on.*
The Application Builder provides the following controls:

Select Objects allows you to select one or several objects simultaneously. It will be especially useful for aligning elements on the screen.

	Static Control
	Edit Control
	Button Control
	Combo Box Control
	List Box Control
	Check Box Control
	Frame Rect Control
	Bitmap Control
	Bitmap Button Control

Please review the detailed information on every control in the Screens Controls chapter.

Dimension Bar

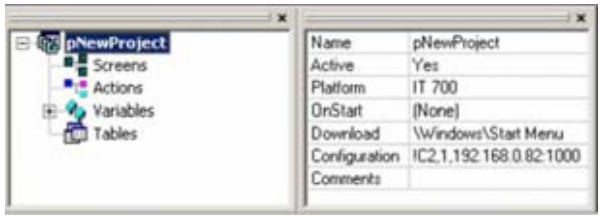
These tools are destined to align and set the dimension of object or controls. Prior to using them, it is necessary to select the objects you want to work in. You can select the objects with the help of the Select Objects button,  , or by clicking each of them while holding down the Shift key. The buttons have the following functions:

	Align the left edges of the selected controls with the dominant control
	Align the right edges of the selected controls with the dominant control
	Align the top edges of the selected controls with the dominant control
	Align the bottom edges of the selected controls with the dominant control
	Resize the selected controls to have the same width as the dominant control
	Resize the selected controls to have the same height as the dominant control
	Resize the selected controls to have the same size as the dominant control

The Project Structure

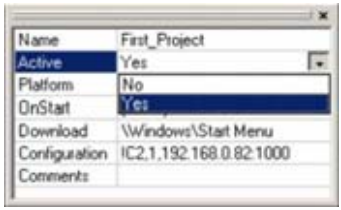
Project

The first step in the application development is to create a project. project is the main object that comprises all the items and program logic. To create a project, click **New** button or select **New** item from the *project* menu. The **Item Bar** will show a new project named *"pNewProject"* by default, and the **Property Bar** will display a table specifying all its properties:



Picture 3: Item Bar and the Window of Projects' Properties

Specify the name of the project in the **Name** field. Please ensure that the name of the project is a proper name and will be the final name of the application that will be sent to the PDA.
The Application Builder allows simultaneous opening of several projects. All the commands however will be performed only for the active project. *(Note: Only one project at a time can be active)*. Select **Yes** from the **Active** field to set a project to the active state.

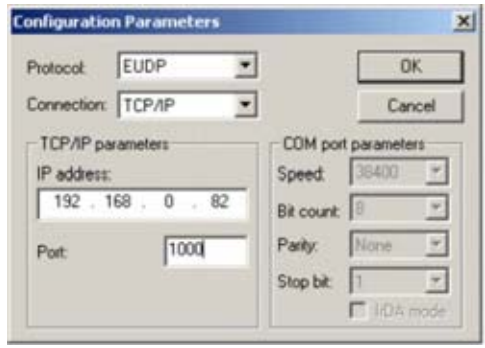


Picture 4: how to make the Project active

The application can work with different models or types of PDA. It is obligatory that you specify in the **Platform** field the type of the device you want to work with and for which the application is destined**.** Thus, you choose type from the pop-up menu.
For the **OnStart** field you can select one of the screens, controls or actions that are in the Project. Then, the program will start by executing an application on the selected Object. If you don't specify anything in this field, the Application Builder gives an error report when generating the application.
In the **Download** field you have to define the PDA directory where the program will be registered and executed while downloading the application. Please, note that in this field you specify a directory located on PDA and not on the host PC. By default the file will be saved in the directory *\Windows\Start Menu directory.*
In the **Configuration** field you specify parameters of the connection between the generated application and the Communication Server.

Note: *Commands specified in the Configuration field are compatible only with the Communication Server.*

Parameters of data transfer are specified in the *Configuration Parameters* dialog box. These Parameters of transfer concern data transfer protocol and type of connection (IP computer address, to which connect PDA, port number or parameters of the COM port connection (all depend on the type of connection).



Picture 5: The Configuration Parameters transfer

Protocol

Several protocols of data transfer are available :

1. PointToPoint
2. UDP - *User Defined Protocol*
3. EUDP - *Enumerated User Defined Protocol*

PointToPoint: is the simplest protocol that doesn't use control character. The data blocks division is done by a *timeout*.
UDP: is point to point protocol with an additional control character (check sum) and a start and end of block character.
EUDP: is the most stable protocol. By its functions this protocol corresponds to the *UDP* but with a block number. This protocol is recommended and defaulted into the dialog box **Configuration Parameters**.

Connection

Depend on type of connection. The most versatile is the Ethernet TCP/IP protocol.

TCP/IP Parameters
With TCP/IP, you have to specify the IP address of the computer to which the connection will be made and the number of the logical port for the connection to Communication Server.

COM Port parameters
If you use one of the COM ports for the connection , values such as *Speed, Bit count, Parity, Stop Bit, IrDA mode* (for data transfer via infrared port) must be identical with Communication Server setting.

***Note**: the transmission parameters can be modified while the application is running with the help of an Operation Configure Action. See operations description in this User's guide.*

Comments field allows to introduce comments to the projects. We'll see how the comments can be linked toa screen, action or control. They are provided in addition to the Application Builder standard information. They will be helpful for understanding all the structure of project, and it's advisable to use comments as often as possible.
The project consists of objects which you can see when you open the project. There are several types of objects: **Controls** situated in **Screens**, **Actions**, **Variables** and **Tables**. (Picture 6).

Picture 6: Components of the Project

Screens

Screens are the base of applications. The different screens of an application form the user interface. The **Controls**. Will be fixed on these screens. The project usually has several screens. To add one or more screens to the project, right-click on the mouse anywhere on the **Item Bar** and select items **Insert =>Screen** (Picture 7) from the pop up menu. An empty screen will appear in **WorkSpace** and in **Item Bar**.

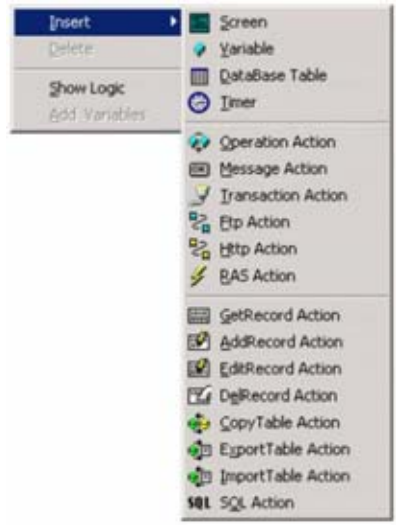


Figure 7: Menu Insert\Screen

A new Screen is associated with a property list, that is displayed in the **Property Bar**. You can modify the by-default values of properties :

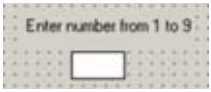
Figure 8: Screen's Property Bar

The **Name** property : a name is given by default. It must be unique in an object family. You can name your project as you want, but for convenience we advise you to make the first character of the name of an object compatible with its type. For example, start each screen name with an "s", "b" for "button", etc. It can makes no difference for a small project, but when you have plenty of objects in your project it can be quite useful to sort things out.
In the **SizeX** property you indicate horizontal dimensions of the screen in pixel. Idem for **SizeY** for the vertical dimension. You can change the color of the screen background by setting it in the pop up menu of the property **BkColor**.
Comments is a free space designed by developers for making comments concerning the Objects functions. It facilitates the comprehension of project during the next reading.
When creating the project, the Application Builder generates the framework of the application. Now, it's possible to add essential elements to the construction of the application
On the screen we'll start by adding controls. To add them, select the required control from the tool bar and then click on the location on the screen you want to fix the control. You can easily change the Object's positions by dragging it with the mouse. When you select one control on the screen, its properties appears in the **Property Bar**. Later in this guide we will review properties

Screens Controls

Static Control

Static control is designed to display a static text on a screen. It can also be used to identify a Data input field and to provide the operator with pieces of information on the data to capture in an Edit Control window (Picture 9).



Pict. 9: Example of static control usage

Description of **Static Control** properties:

Property	Description
Name	The unique name of the control in current screen.
OriginX	The left position of the rectangular control in pixels.
OriginY	The top position of the rectangular control in pixels.
Width	The width of the rectangular control in pixels.
Height	The height of the rectangular control in pixels.
Data	The data of the control. The data can be modified by some Action.
FitToHeight	Specifies if the height of the text font depends on height of the rectangular control . If the value is set to 'Yes', the font height is equal to the rectangular control height, if 'No' - the font height is equal to the standard text font height.
Color	The color of the text.
Comments	The control description and comments.

To modify the control position you can :

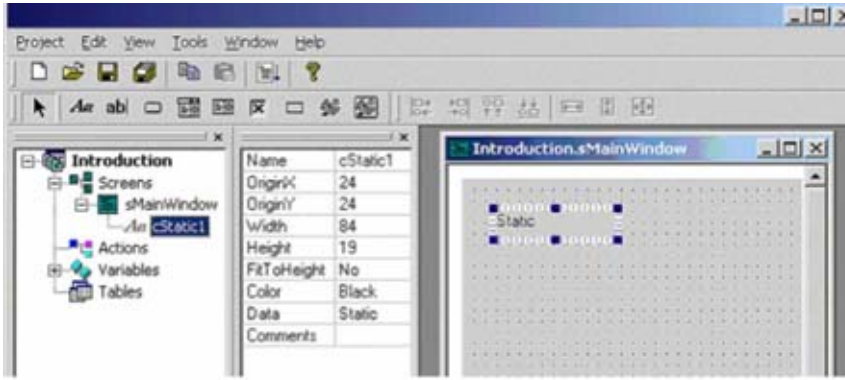
1. Modify the parameters values directly in the corresponding properties
2. Select the control and use the drag-and-drop mode.

When you aligned the text it's easier to use the tools destined to these functions. They are in the **Dimension Bar** :

Note: *While modifying the size of window, you don't modify the size of characters. If you want to link the character size to window height you have to select "**Yes**" in **FitToHeight** parameter. If the width of characters exceeds the one of the window, thus only the characters in window will be visible.*

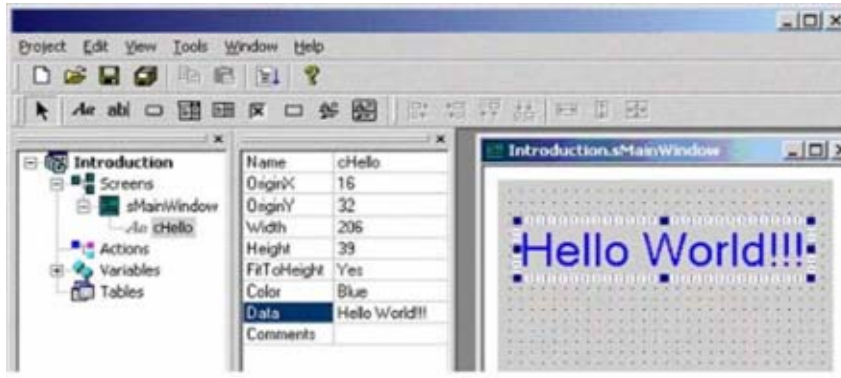
Let's create a program that will display "***Hello World***" on the screen of your PDA. Open a new project and name it "*Introduction*". By default, replace, in the Property field, in the property **Name,** the name of your choice : "*pIntroduction*". Then, insert a new screen (by right-clicking on the **Item Bar** and then selecting **Insert\Screen**). You can also change its default name : *sMainWindow*. Now, insert Static Control : click on in **Palette Bar**

and then click the place you wish on the screen or click the right button on the screen and select Insert=>Static.
The result must be identical to the picture 10 :



Picture 10: the main window with the Static Control added

Change the position of the Control. In **Property Bar**, for the property *Data* you enter the text to be displayed on the screen. The default text is "*Static*". The value to enter in this Project is "Hello World". Once your new text value is validated using the Enter key, it appears on the screen. You can leave it as it is or change the font size as well by setting the value to «Yes» in the **FitToHeight** field. You can change the size of the window by using the resize handles or change the character font color by using a new value in the pop up menu. Rename control on "Hello". After all the above-mentioned operations have been performed, the main window will look as



Pict. 11: The program's main window with the changes made in the Control Property Bar

Now you have to indicate the screen where the application is to be started in the Project's **Property Bar.** Our example presents the unique screen that you have to specify in the *OnStart* field

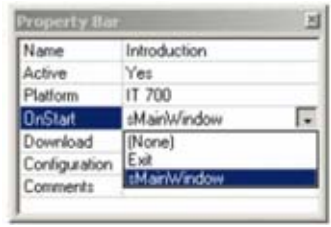


Figure 12

In **Platform** Project's property select the model of PDA you will load the program to.

Note: *If the model is not reflected in the menu, you always have the option to select another one from the same category (same processor type and same screen size). If your model is in the menu you'll be sure that the options will be supported by your application (radio WiFi, bar code reader, etc…)*

Save the project and click on the **Generate** button. Check the message in the **Output Bar**, if you see the message **"0"** error(s), **"0"** warning(s) message, it means that the generating phase of the program has been carried out correctly, otherwise the program will report in **Output Bar**. With this command the program will be sent to PDA. Now open Start Menu on the portable terminal, there you can find the
Newly created program and start it (double-click on the program name). «Hello world!»

Figure 13 : the application is launched on the PDA

Edit Control

Edit Control (fig.14) : allows a user to type data text. Edit control contents can be processed by application and modified by various Actions.

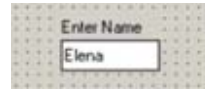


Figure 14: The example of Edit Control usage

Like Static Control, Edit Control has its own name that is specified in the **Name** field. In the **OriginX**, **OriginY** fields you set the horizontal and vertical in pixel position of the rectangular's top left corner working with the Control window. The **Width** and **Height** fields fix the size of the text area. In the **Data** field you can type some characters that will be displayed in the data input area when the application starts (for example, it could be a point line to materialize the place of characters entered by the operator). Or you can leave it empty.
Validation can contain one of two values: "*DataType*" or "*Template*". To select *DataType* means that the type of data that will be allowed while the application is in operation.is entered in the associated property **DataType** (Figure 15)

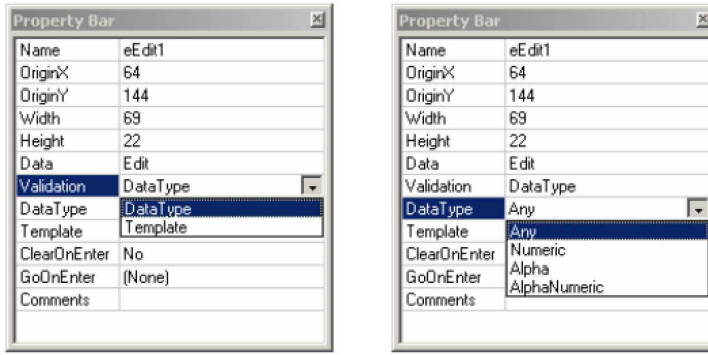


Figure 15: Possible options for Validation and DataType

Please note that if you try to type characters whose type does not correspond to the one you have specified in the **DataType** field, the program will not report any error, but these characters won't be accepted and displayed on the control window. If you select "Any" in the **DataType** field, you will be able to enter any of the admissible types of data. If you choose "*Numeric*" only numeric characters will be accessible. "Alpha" – the entered data can consist of Latin alphabetic symbols only (from A to Z, and from a to z). "AlphaNumeric" – the line entered can only consist of Latin alphabetic and numeric symbols.
If the restriction to the type is not adequate and you have to specify the structure of the entered data more accurately , use the Templates. To set the template, firstly select *Template* in the **Validation** field, and then specify the pattern of the data.
The rules to structure a *Template* are :

Symbol	type of characters
#	A number from 1 to 9 (numeric value only)
%	A lowercase or uppercase letter of Latin alphabetic (A-Z,a-z)
?	No matter which character

*	No matter which line of arbitrary length

For example, template **##.#?%%** can correspond to the data of **45.2-di** or **21.2=Ab** and the like. When the template is specified, the user can enter only the data that corresponds to the pattern.
ClearOnEnter : If you want **Edit Control** to become empty when you jump to it from any other Control, screen or another element, set "*Yes*" in the **ClearOnEnter** field. This function allows you to reduce the number of operations performed and is widely used by data scanning. ***Note:*** ***Data*** *property can be empty.*
In the **GoOnEnter** property you specify an action to be executed right after the **Enter** key is pressed or laser scanning is done.

To get a better idea of **Edit Controls** and their capabilities, let's add them to our "Introduction" project :
The first control will serve for the input of a string of any numeric or characters type. The second will accept only numeric characters. The third will verify the data format entered according to a **Template**.
Open the existing project. To display the screen's contents in the Workspace, open **Screen** folder and double-click on its icon. Then place an Edit control on this screen. Since it is meant for data of any type, let's name it "eAny". Delete all on **Data** property so the field is blank when the cursor is positioned in this window.
In the **Validation** field, select "*AnyType*" to authorize any typed character.
Let's set *"Yes"* in the **ClearOnEnter** field. It means you do not have to manually delete the information entered earlier. Let's add the second control for numeric data input field and name it *"eNumeric"*. Like in the first case, choose *"Yes"* from the **ClearOnEnter** field to delete its contents. In the **Validation** choose the *"Numeric"* type. In the **ClearOnEnter** field select *"No",* for comparison with the earlier Control and note the difference. Now its time to create the last control where we wanted to use *"Template"* model. Let's name it *"Template"*. Select as type *"Template"* in the **Validation** property, and then define the Template. For example, you want the data inputs to begin with two numeric characters, then "dash" and letter. Consequently, enter such "##-%" string in the *Template* field: # is to define a numeric, "-" is as character and % is to indicate the letter. Set **ClearOnEnter** field to *"Yes"*. Now go back to the first control and select *sMainWindow.eNumeric* from the pop up menu of the **GoOnEnter** field. It means that after you have entered the data in the text field and pressed the **Enter** key, the control will be transferred directly to the denoted field. For this example it will be on the *eNumeric* Control of the *sMainWindow* screen. Repeat this procedure with the second control. With the third, set the back to the first control. As a result, you will have a sort of cycle going from one control to another.
If you want anyone else to work with your application, you have to ensure your program's interface is easily understood. If you add a data input field, you must tell the user what exactly he has to enter in this field. Therefore, let's add to our project several static controls that serve as hints and tips.
The only thing you have to do is to click on the **Generate** button to generate the application. If everything
is done correctly, you will see *"The program has been downloaded to the device"* message in the **Output Bar**, otherwise the program will report to you of where an error occurred.
Go to the Start Menu and start your updated program (Figure 16)

Figure16: The application is launched on PDA.

Now, apart from the "Hello world!" phrase you can see three data input fields on the screen. Try to enter some optional characters in the first field. In the second you verify the only accepted input numeric and the third accepts data corresponding to *Template* model. Also do not forget to pay attention that when you jump from one field to another, the first and the last fields are cleared, but the intermediate one keeps the input data.

Button Control

Button control is an object that allows you to execute an Action by clicking on the button while the application is running (Figure 17)

Figure 17: example of Button Control usage

Property Bar of the **Button Control** consists of standard fields that are common to all controls, such as *Name, OriginX, OriginY, Width*, and *Height*. In the **Button** field you specify the inscription on the button. Some action is to be executed when the button is pressed (otherwise it becomes useless). This action is set in the **OnPress** field by selecting it from the list box. The button becomes active or passive when the program starts depending on whether **State** field is set to the "*enabled*" or "*disabled*" mode. More details on this property will be given in the **Operation** chapter.

Bitmap Control

We have reviewed controls dealing with text data. It's also possible to display graphic data on screens. This capability will be also used to illustrate a button with a graphic figure (see **BitmapButton Control**).

Bitmap Control is an object designed to show a graphic picture on the screen. With its help you can place a given bitmap in any part of the screen. (Figure 18):

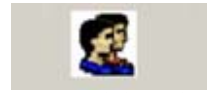
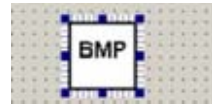


Figure 18: example of Bitmap Control usage

Its **Property Bar** also contains *Name, OriginX, OriginY* fields, that are standard and you already know their functions. Apart from these ones, there is also a special **File** field. It is used to give the path to the bitmap you want to insert.
This object is an easy-to-use one. Add it to the active project and change its original name to an detailed one. If the following picture appears on the screen. (Figure 19) … :



Picture 19 This is what appears if the File name is not specified in the Bitmap Control properties.

… the reason can either directory is not the right one, or the file is not in this directory.

BitmapButton Control

BitmapButton Control is an object that combines Bitmap and Button Controls properties. In fact, it is a button with a graphic image (Figure 20)

Figure 20 : Example of button with graphic image

The Control's **Property Bar** contains a combination of fields appropriate to Bitmap and button controls. You are already familiar with them.

To understand the button usage let's study the following example.
Create a new project and name it "Graphic". Add two screens : "sFirstWindow" and "sNextWindow". Select *"White"* from the **BkColor** field for the first screen, and any other color, for example, *"Blue"* for the second one. Jump to the sFirstWindow screen and add a button to it. In the **Button** field type "Next" and change **Name** of this button to "*bNext".* So far, this button cannot yet operate. To make something happen when it is clicked you have to assign to it some Action. We will order it to transfer the user to the next screen. You can do this by selecting "*sNextWindow*" from the **OnPress** field's list box. (Figure 21)

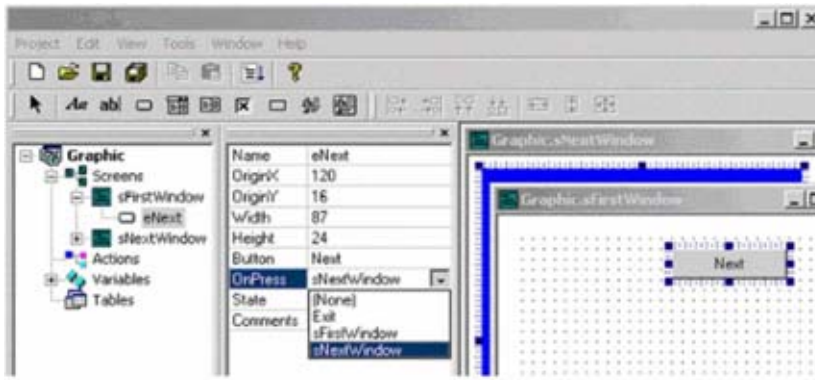


Figure 21: setting of sNext button

Now go back to the *sNextWindow* screen and insert an image. You can do this by clicking **Bitmap button** in the **Palette Bar**, or by right-clicking on the mouse button anywhere on the screen and selecting *Bitmap* from the **Insert** menu which appears(Figure 22).

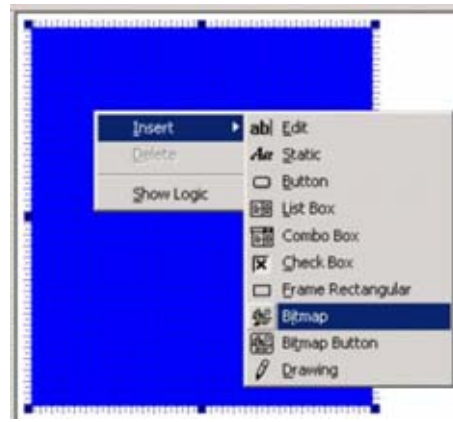


Figure 22 : a Bitmap Object adding..

A square with ”BMP” on it has appeared on the screen, as well as its property table on the **Property Bar**. Name it “bPicture”. Let’s use images that come with the product. Choose Open from **File** menu. Go to the folder where the Application Builder is installed and find **Logo** where you’ll find images supplied with the product. Select “User”.

Figure 23 : a BitMap selection

After the bitmap appears on the screen you can easily move it by dragging it with the mouse. Let's add to the screen a button with a bitmap that will transfer you back to the *sFirstWindow*. (Figure 24).

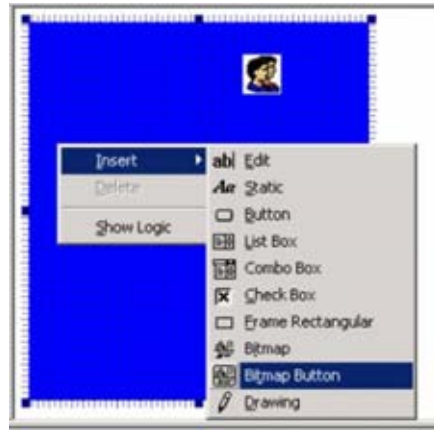


Figure 24 : a Bitmap Button append

Thus, a new **Bitmap Button** control has been added to the active project. Name it "bFirst". As in the previous case, choose **Open** from the **File** menu and go to the Logo folder. Select a bitmap named "Quitter".

Let's assign this button to transfer you to the previous screen. You can do this by clicking its "sFirstWindow" name in the **OnPress** field. Now you have to set which screen is to appear on your terminal when the program starts. Do this by clicking the project in the **Tool Bar** and selecting one of the screens, for example, "*sFirstWindow*" from the **OnStart** field of the screen's **Property Bar**. Click **Generate** and after the project is saved you can test the project developed directly on the PDA. (Figure 25).

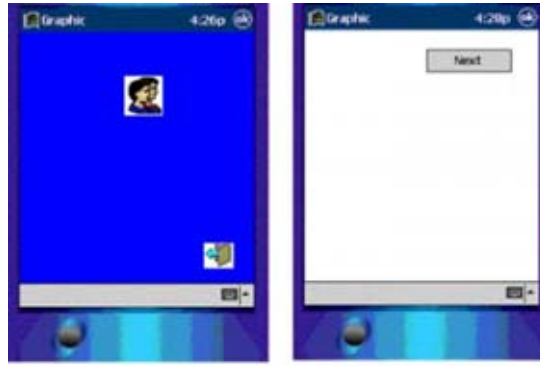


Figure 25 : a button to pass from one screen to another

Check Box Control

Check Box Control displays a text string with a small check box to the left. An example of **Check Box Control** is given below.

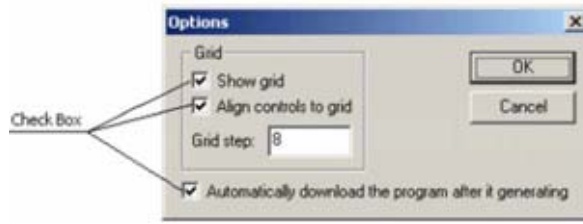


Figure 26

Apart from the standard properties, Check Box Control's **Property Bar** has a **Check** field. Everything that is written in this field is displayed to the right of the check box. Any box can return a value when running the application. The marked check box control sends back value 1 and the unmarked one value 0.

List Box Control

ListBox Control is dedicated to display a list of items.
Property Bar : if you open the **ListData** dialog box of **Data** property and enter data in, they will be displayed in List Box while the application is running (Figure 27).

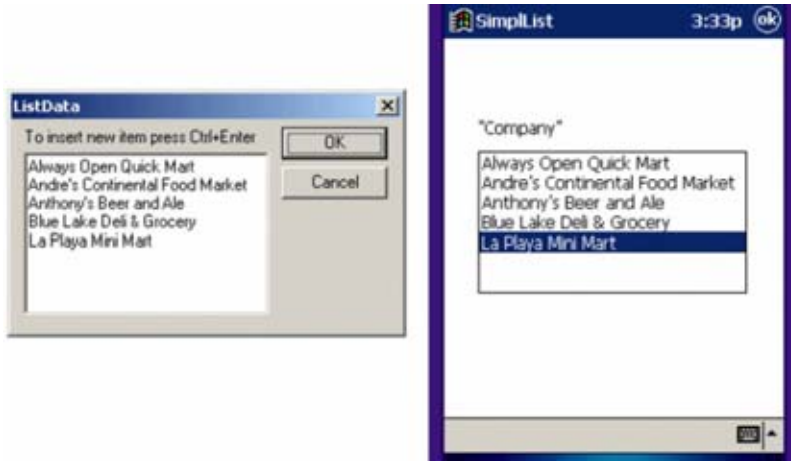


Figure 27 : ListData contents are displayed on the screen.

To allow the application to delete, modify data from the List Box or add new data, or assign the new data value from the box to an other variable control, you have to use special operations. We'll see Operations on the Action chapter.
In the **OnDoubleClick** field, specify what action is to be executed when the user double-clicks the selected record with the stylet.

ComboBox Control

The **ComboBox Control** is related to the **ListBox.** The difference is that **ComboBox** allows you to enter data on the top zone. This object will give you the option to either select an object in the list, or enter data from the keyboard.

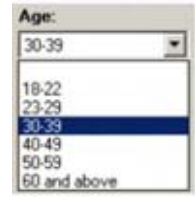


Figure 28 : the ComboBox usage

You will find the same fields in its **Property Bar** and will be able to add data to the **ListBox** or remove them from it in the same wayas with the **ListBox** control. The return value of this control is a string specified in the data input field. You can add data to the **ComboBox** Control, in the same way as with the **ListBox Control**, with the help of the *Assignment* action and delete them using *Reset*. More details on all operations will be given in the **Operation Actions** chapter.

FrameRect Control

FrameRect Control is dedicated to outlining or dividing groups of elements by drawing a frame, a box or a line on the screen. It can appear as a string or frame of rectangle.

Figure 29 : examples of frame usage

Drawing Control

The **Drawing Control** allows to draw characters or graphic items using stylet and the sensitive screen and saving or deleting them. (Fig.30)

Figure 30

To save or delete the item, use the **Operation Action SaveDrawing** or **ClearDrawing**. To allow the capture of a drawing, a drawing window has to be defined on the screen. This function is performed by a **Drawing Control**.

The **Drawing Control** is available from the **ToolBar** : Click on  icon, and place it on the screen.
This control has the following properties :

Property	Description
Name	The unique name of the control in current screen.
OriginX	The left position of the rectangular control in pixels.
OriginY	The top position of the rectangular control in pixels.
Width	The width of the rectangular control in pixels.
Height	The width of the rectangular control in pixels.
Directory	The device directory to store the image file in.
File	Defines the file to store the drawing in. The saving is performed during execution of **SaveDrawing** sub-action of **Operation Action**.
Comments	The control description and comments.

Variables

Variables are designed for temporary data storage. The **Variables** are stored in memory under a variable name. The name is used to retrieve information from memory. For example **Variable** « *x* » with value 5. If we execute expression 10+x the result will be 15. During program execution, the **Variable** value could be modified. The **Variables** are of different types : **Numeric** , **Data String ,** or specific: **Data&Time.**
There are two standard **Variables** : *False* and *True* . They are used in diverse expressions. The **False** value is equivalent to zero, and **True** value is 1. Those values are convention, that means, it is advisable not to modify them :
The following table shows all **Variable** properties and their description.

Property	Description
Name	The unique name of the Variable.
DataType	The variable type. Can be one of the following: String, Integer or Date Time.
Data	The initial value.

Date Time type Variables :
With this Variable type you can return date-and-time at your request.
When you check the Date Time setting, enter the day (time) value in the **Data** property. This is the format of your Date Time **Variable**.
To get date and time, you should use a combination of specifiers, described below.

Specifier	Description
d	Displays the day as a number without a leading zero (1-31)
dd	Displays the day as a number with a leading zero (01-31).
m	Displays the month as a number without a leading zero (1-12).
mm	Displays the month as a number with a leading zero (01-12).
yy	Displays the year as a two-digit number (00-99).
yyyy	Displays the year as a four-digit number (0000-9999).
h	Displays the hour without a leading zero (0-23).
hh	Displays the hour with a leading zero (00-23).
n	Displays the minute without a leading zero (0-59).
nn	Displays the minute with a leading zero (00-59).
s	Displays the second without a leading zero (0-59).
ss	Displays the second with a leading zero (00-59).
t	Displays the time hh:mm:ss
c	Displays the date dd.mm.yyyy

Example: if you write "yyyy/dd/mm", in runtime mode the result of this variable will be the current date - 2000/27/11.
***Note**: please, refer to the example made after the review of operations to get an overall idea of how to use variables of the Data&&Time.*

Prior to addressing a variable you have to name it by taking the following steps:

1. Make sure that the project you want to add a variable to is active;
2. Click right button on the *Item Bar* and then click in the pop up menu **Insert\Variable**.
3. Change the standard name.
4. Enter an initial value.

Tables and Databases

The application's goal is to work with data. If you want to manipulate data with the help of the application, you must first create a database or table to store the data. Tables are stored in databases. In a table object we can define a table structure and its location. While the application is running, it checks if the defined table exists, and if not, automatically creates it. So, normally, you do not need to care about creating tables, you will just have to define its parameters and structure.

The application works with tables by using Microsoft® CE Data Access (ADOCE). ADOCE is part of Microsoft® Windows CE and define interface to work with database Win CE. The data type depends on Provider Driver. There are two types of database that can be used with Windows CE : you use database supplied with Windows CE or you set up to your PDA on another server of the database, for example Microsoft SQL Server for Windows CE, Oracle for CE or any other type of database compatible with CE. The data type has to be indicated in **Property Bar**. Then you will be able to transfer data by coping tables to the host computer using the **CopyTable Action** or send database request using **Transaction Action** in connection with a Communication Server.
Note : note that the application automatically creates tables only for the Windows CE database. If you want to work with others databases, you should create them manually with the help of the SQL Action.
As you can see on the picture below, a **Table** is just a list of records. Every entry contains several fields where the data are stored.

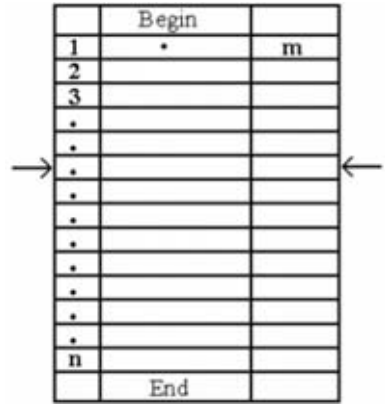


Figure 31: Database table

The structure of the table as well as the parameters of its fields are defined in the following **Property Table** :

Property	Description
Name	The unique name of the action.

<table>
<tr><td>Provider</td><td>The database connection string. It depends on the database type to work with. There are several options that are shown in the following table.
<table>
<tr><th>Provider value</th><th>Database type</th></tr>
<tr><td>Empty string</td><td>The table is located in the database of Windows CE. It requires a minimum disk space but is quite slow. It is similar to text files and is recommended for simple applications.</td></tr>
<tr><td>Path to the .cdb file, for example: \Windows\test.cdb</td><td>The Pocket Access database. The medium solution, it is recommended for common applications.</td></tr>
<tr><td>Provider depending string. About the syntax, see database provider documentation.</td><td>The external provider database. Usually, it is a faster database than predefined ones. For example, Microsoft SQL Server for Windows CE is several times faster than Pocket Access database, but also requires more disk space. It is recommended for advanced applications.</td></tr>
</table></td></tr>
<tr><td>Table</td><td>The database table name.</td></tr>
<tr><td>Parameters</td><td>The database structure. It is specified in the table dialog box.</td></tr>
<tr><td>Comments</td><td>The action description and comments.</td></tr>
</table>

Note : it's very important to understand the meaning of **Provider** property. The type of used table depends on the value of this property. If the value is empty, the program creates the table in the shape of a text file that minimizes the place to memorize the information. According to the case with PDA, the disk space of the Flash memory is limited in its size. But this file type has slower access, especially when you're looking for data in the table. This type is for simple applications and for large data volume.
If you need a faster access to data, you'd better use **Pocket Access** databases. You can do this by specifying in **Provider** property the path to the PDA directory where the table will be registered as well as the name of the file with **.cdb** extension. For example: **/Windows/test.cdb.** We recommend this mode for the more powerful applications as it presents the optimal balance between access speed and database size. The use of specific databases is justified for the applications that make complex transactions on very important data volumes.
In the **Table** property, indicate the name of table that will be created on PDA. If you work with files on **Pocket Access** format (**cdb** extension), the table mentioned is saved as indicated in the **Provider** property. This directory can contain several tables.
In the **Parameter** property is determined the database structure. When you select this property, a button with three points is displayed in the right side of the window. When you click on this button, the table dialog box appears and allows you create the structure of the Table.
From this dialog box, you can add fields to table (New button), delete them (Delete button), alter their positions in relation to each other by clicking Up and Down , specify their name in the *Name* field, define their contents in the *Type* field. There are two types: "Text" and "Integer". To work with the text type you have to specify the maximal number of symbols in **Length** property (Fig.32).

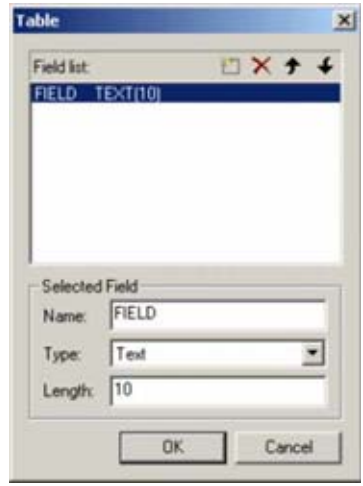


Figure 32: Dialog Box of table structure

We also recommend you name the fields with meaning names in order to avoid problems of referring, in case you make links between input data through operator and their memorization in tables and corresponding fields.

To create a new table, do the following steps :

1. Make sure that the project on which you want to add the table is active;
2. Right-click on the **Item Bar** and click in the pop up menu Insert\Database Table. The new table has been added to the active project, and the **Property Bar** shows the properties of this table .
3. Change the name of the table as suits you.
4. Define the type of the table in the **Provider** field.
5. Define the structure of the table in the **Parameters** property.

Actions

An **action** is a set of operations performed by the application on a given event.
You can for example create an action and assign it to run on button click or on the end of another Action. Depending on the result of executed action the program will go on Actions defined in **OnOK** or **OnError** properties. **OnOK** means that no error occurred while the action was performed, whereas **OnError** is activated in case of some error occurs. Accordingly, you can define a sequence of actions that will compose a part of the application logic.

Message Action

Message Action is to display a message in a window. It is often used to inform the user that some error has occurred while the program is still in operation. The way message will look is defined with the help of the Property table, where you can specify the name of this action in the **Name** field, text of the message in the **Message** field. Select one of two types of window in the **BoxType** field (Fig. below): table's type "*OK*", or "*Yes/No*".

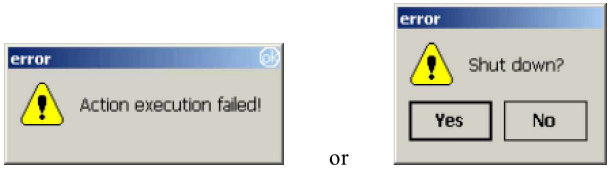


Figure 33 : message actions

In the **OnOK(Yes)** field you have to define, what action is to be performed after the user clicks "*OK"(*or YES for the second type).The **OnNo** field is reserved for the *"Yes\No"* type. There you also have to define what the program has to do if the user clicks the "No" button.
To create a new message window, do the following steps:

1. Make sure that the project to which you want to add the message is active;
2. Right-click on the **Item Bar** and then click in the pop up menu **Insert\Message Action**. A new message action has been added to the active project and the **Property Bar** shows the properties of this table.
3. Name this Action.
4. Define the text message information.
5. Select the type of the table and assign an action to its buttons.

How to Work with Local Databases

There are several basic objects that you will constantly use to access databases. These objects can be classified in 3 groups :
Non-visual : *AddRecord , GetRecord, DellRecord, EditRecord, CopyTable* and *SQL Action*
Visual : *Static Control, Edit Control, ListBox Control, ComboBox Control*
Linking : *Operation Actions*
The first group comprises actions designed for manipulating tables.
The second group consists of visual actions that show data to the user and allow the modification of editions or input.
The third group of actions helps to link the two earlier object types.

In the present section we will learn how to work with local database using the following Actions : *AddRecord*, *GetRecord*, *DelRecord* et *EditRecord*.

AddRecord Action

AddRecord Action gives you the option to add data records to a local database (running on the PDA terminal). This action adds a new record to the end of the table. AddRecord Action's properties consists of the following fields: **Name** is designed to assign the name to an Action. This name must be unique in the same project. It calls this action in the project. In the **Table** field you define the type of the table where data will be recorded.
Clicking a button, you open a dialog box that displays all the fields of the in question table. The goal of this operation is to determine the sources of information that will be memorized in the table. This operation defines the links between the Edit Controls and the fields of the table. This link is performed in graphic form by using for each field a corresponding control in the list. This control can be a variable or an Edit Control (Fig.34)

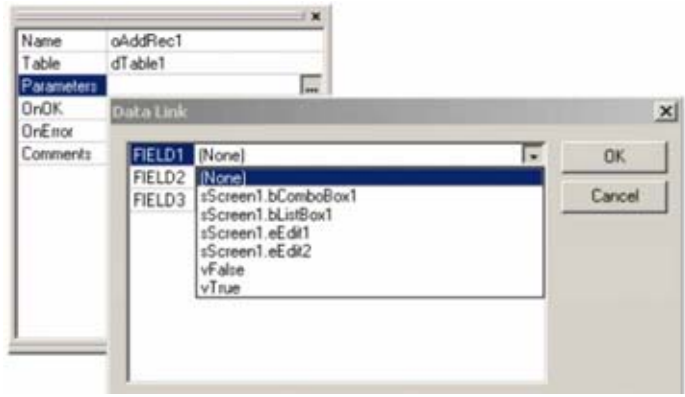


Figure 34 : the dialog Box of Parameters property

Description of the **AddRecord Action** property :

Property	Description
Name	The unique name of the action.
Table	The table name associated with the action.
Parameters	This action's link parameter definition .
OnOK	Defines action to which the application goes if no error occurs while performing the action.
OnError	Defines action to which the program goes if an error occurs while performing the action.
Comments	Comments on this action

If no error occurs while the Action was performed, the program will go to the Action defined in the **OnOK** field. In the **OnError** field you have to define an action to be performed if an error occurs.

To create a new **AddRecord** Action, do the following steps :

1. Make sure that the project to which you want to add the Action is active;
2. Right-click on the Item Bar and then select in the pop up menu **Insert\AddRecord Action**. As a result, this new Action has been added to the active project, and the **Property Bar** displays the properties of this new action.
3. In the property **Table** select the Table to which you want the data to be added.
4. Specify the links between the fields of database and controls from which the data will come.
5. Specify actions to perform the **OnOk** and **OnError** events.

GetRecord Action

GetRecord Action allows to retrieve and filter data from the database and assign them to the application's objects, variables or controls.
The **Property Table** below describes action's properties :

Property	Description
Name	The unique name of the Action.
Table	The database table to execute the Action on.
GoToRecord	The value can be set to *First*, *Next*, *Previous*, *Last*, *Query* or *Query To List*. The *Query To List* entry is designed to select and copy data from table fields to a List Box or Combo Box controls. It is a stand alone action and no any additional operations are needed. Before sending data to the box, it must be cleared. The data links are defined in the **Data Link** dialog box of the **Parameters** property. The *Query* entry is used to select or filter records to work with. It is utilized in conjunction with moving entries (First, Next, Previous, Last). These entries scroll throughout selected by Query entry record set. The Query filters only desired records and set pointer to the first record that satisfies filter conditions. If the **Query** property is empty, all database records are selected and pointer is set to the first table record. If you work with database table first time, you must start with *Query* entry and only after that use moving entries. The data links are defined in the **Data Link** dialog box of the **Parameters** property.
Parameters	This property is linked to the **GoToRecord** property only when **GoToRecord** is set to *First*, *Next*, *Previous* or *Last* (useless in other cases). Through this dialog box : you can link database fields with controls or variables of the application.
Query	The **Query** is used to filter result record set. The filter is specified in the query dialog box. This property is valid only if the **GoToRecord** property is set to the "*Query*" or *"Query To List"* entry.
RecCount	The returned records number of the **Record Set**. It's valid only if the

	GoToRecord property is *"Query"* or *"QueryToList"*.
OnOK	Defines Action to which the application goes if no error occurs while performing the *GetRecord* action.
OnError	Defines Action to which the program goes if an error occurs while performing the *GetRecord* action.
Comments	The box reserved for comments on this action

Figure 35

To be able to work with *GetRecord* Action of *First* type (*Next, Previous or Last*), you have to perform the *GetRecord* action using **"***Query***"** in the **GoToRecord** field. The returned value of this action will be the first record (Next, Previous or Last) from the **Record Set.**
If you need to display the complete Record Set, you first have to click "Query To List" in the **GoToRecord** field.
Thus, the record set depends on the searching text specified in the **Query** property.

Example : (Fig. 36)

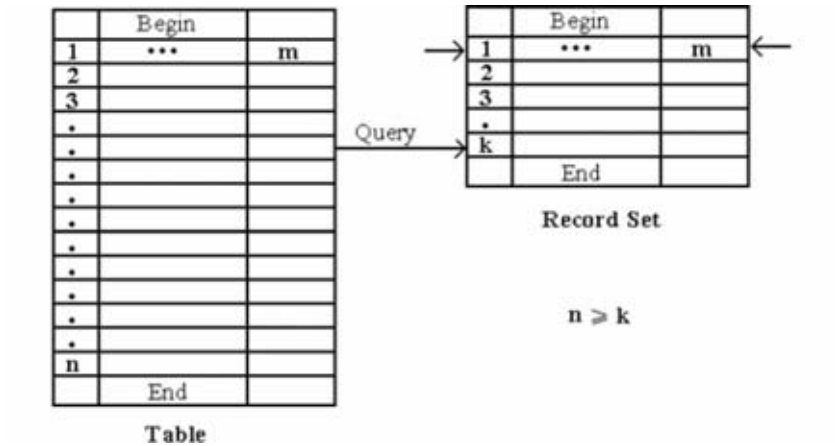


Figure 36

The returned value of action with first parameter will be the first one of record set. The returned value of the action with the parameters Next, Previous and Last will be respectively the

following, preceding and last record. To show on the screen the contents of Record Set, choose "Query To List" in the **GoToRecord** property.
Note: *After the sub-action with "Query To List" has been accomplished and before use the First, Next, Previous or Last Action, you have to use again "Query" Action.*
Note: *a try to pass to the previous record if it's on the first one, or to pass to the next one if you're on last one, will be considered as a program error.*
In the dialog box of the **Parameters** field you have to define which control or variable the value of data field will be assigned on. (Figure 37).

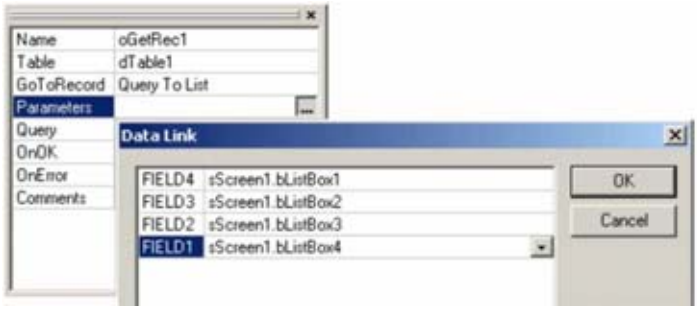


Figure 37

Sometimes, you don't need to extract all records from database. The Application Builder will assist all your requests. Thus, you just define the criteria according to which the table records will be filtered. The selection criteria are specified in the dialog box of the **Query** field. (Fig. 38).
This property is used only if *"Query"* or "*Query To List"* values are set in the **GoToRecord** field.

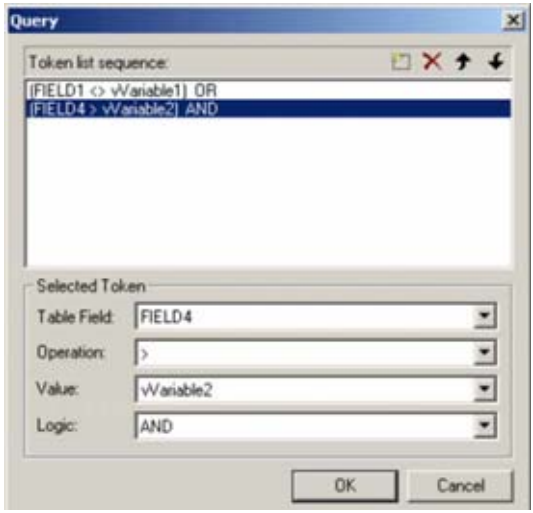


Figure 38: Dialog Box to formulate a demand

Every criteria presents a sort of condition where the value of Data field is compared to the value of the control or variable. The contents of the request consists of informative fields of **Selected Token** (Fig. 38).

Table Field	Indicates the data field, the search has to be executed on
Operation	Indicates the type of operation that will be done between the **Table Field** and the **Value**
Value	Indicates the control or variable the value of which will be used in the operation
Logic	Logic operator that stands between the different conditions: if the condition is the last, the program will ignore the logic operation. You can delete parts of the query, modify them, and add new ones with the help of the buttons located in the top right corner. Their usage is identical to those of the table's dialog box buttons.

In the list box of the **OnOK** field you have to select an element to which you want the application to go on. This element can be an action that is executed as soon as the data has appeared. Similarly it is indicated, in the **OnError** field how the application goes on if an error occurs while the action executes. In many cases it will be an information message for the user. To create a new *GetRecord* Action, do the following steps:

1. Make sure that the project you want to add the action to is active;
2. Right-click on the item bar. Select in the pop up menu **Insert\GetRecord Action**. As a result, a new action has been added to the active project, and the properties bar displays its properties.
3. Change the by-default name to your own.
4. In the **Table** field, select the table where you will search data.
5. Select the corresponding element from the parameter of the **GoToRecord** property. If you have chosen either "*Query*" or "*Query To List*", specify the filter to appropriate records of the *Query* dialog box (**Query** property)
6. Specify links between table fields and controls.
7. Specify actions to fulfill if an error has occurred or not.

Now we know enough to create a project using these actions. For example, you want to create an application designed to work with a database storing the surnames of the firm's employees. This application will consist of one data input field, a list of database records, and a button that will record surnames to the database and automatically display them on the screen.
Create a new project and name it *"Name"*. Add to the project a screen named "sMainWindow". and Edit Control, Button Control, as well as List Box. In the property field name the Button *"bAdd*", with an *"Add"* description on it in **Button** Property. Select Edit Control active, and change the existing name to "eLasName". Select *"Data Type"* value in the **Validation** field and then set **Data Type** field to *"Alpha"*. As a result, the operator will be able to record letters. Also, click *"Yes"* in the **ClearOnEnter** field, so that the operator of your program will not have to delete the new-recorded surname prior to record a new one.
Let's create a simple database to store the records. You can do this by clicking the right mouse button on the Item Bar and then selecting from the pop up menu *Insert\DataBase* Table. A Table item will be added to your project. In the **Name** field type *"dTableName"*, and in the **Table** field define the name of the table *"DBLastName"* where the recorded surnames will be stored. Now you have to define the structure of the database (number and types of field). One field will be enough for us, therefore, click the button to go properties and the table display. Create a

new field with button, change its name to “LastName”, define the type and set the maximum and minimum length of field (10 for this example, Fig.39).

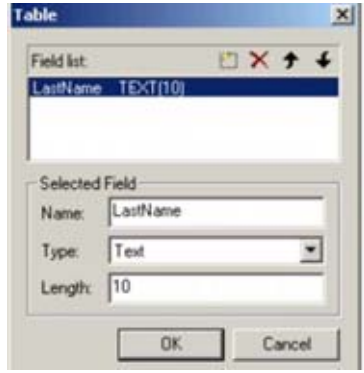


Figure 39: Table dialog Box

The database is created. Now we have to define an action as a result of which new records will be added to the existing *DBLastName* database. Click the right mouse button and select from the pop up menu *Insert \AddRecord Action*. The action has been added to the project. In the table displayed in the **Property Bar**, define its name – *“oAddLastName”,* as well as the name of the table item – “*dTableName”*. (Fig. 40).

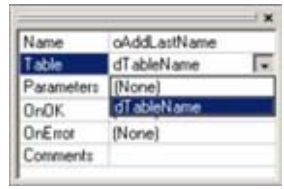


Figure 40 : property field of GetRecord Action

Then, in table you have to define where the data will be recorded. Click the button ... in **Parameters**. The opened **Data Link** dialog box display. Make this box to appear as on the figure below :

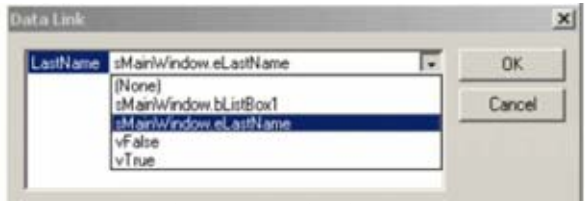


Figure 41 : Data Link dialog box

Select “sMainWindow.eLastName” (name is composed of a screen name that’s placed according the Control name) from the list box and click “OK”.

Now it is time to define an Action that will display the database contents. Click the right mouse button on the item bar and then click Insert\GetRecord Action. Change the original name "oGetRec1" to "oGetAllRec" right away. Just like you did while working with the *AddRecord* Action, define *dTableName* as a name of the **Table** item, where the properties of our database are indicated. Since we want to display the full list of surnames, it will be right to select *Query To List* from the **GoToRecord** field. For this Action you also have to define an item that will send the list of surnames. First, we'll display these surnames in *ListBox*. Name it "bListLastName", and then again mark **GetRecord** Action in the **Item Bar** and get back to its table of properties. In the *Data Link* dialog box of the **Parameters** field click "bListLastName".

It would also be helpful for the user to display a message if some error occurred. So, let's add to our project a **Message** Action, just like we added the previous two. In its table of properties type the name *mMessage*, set the type to "*OK*" and define the text of the message as *"Action incorrect"*. In the **OnOK (Yes)** fields select "sMainWindow.eLastName" – that will be the object the user will be transferred to when click of this button.

Mark oAddLasName Action in the Item Bar and click "*oGetAllRec"* in the **OnOk** field. This will ensure the display of the surname in the list on the click of the *Add* button. In the **OnError** field click "mMessage", and the user will receive error report messages.

What you still have to do is to specify the logic of program's operation on successful or unsuccessful results of the Action for "*oGetAllRec",* as well as to assign an action to the *Add* button. Therefore, in the **OnOk** field select *sMainWindow.eLastName* to determine the cursor's position in the input name field.. This will ensure the return of a cursor to the data input field, and in case some error occurs, the same mMessage will be displayed.

In the **OnError** property select "mMessage". In this case the operator will receive the error message. We have to specify if the logic of "oGetAllRec" action execution of the last one is correct or not, as well, as it has associated with an add button. Thus, select "sMainWindow.eLastName" in **OnOk** field to positionthe cursor. If an error is detected, "mMessage" is necessary: in the **OnError** property select *"mMessage".*

Now, we have to specify the Object to which the application will go first when starting. It can be made in the **OnStart** property, where you select "sMainWindow".
The project is finished. You can generate the code that will execute on PDA by pressing the **Generate** button. If the program displays an error message, verify your program, the Application Builder indicates the type of error is detected in **Output Bar** that's below the screen (see fig. 42) :

Figure 42 : Output Bar

The advancing loading Bar appearance means the application is well generated and the Application Builder transfers it on PDA.

Note: *if application is generated for Microsoft emulator, the advancing bar doesn't display. Execute the program on PDA and enter some values. Your screen has to have the following form:*

Figure 43 : executing programs on PDA

The application allows you to record surnames in the database and display them on the screen – so, we achieved our purpose.
Now, set up the application that allows you to find a specific record. In theory, you already know how to do this. Let's work with our example and add names and the age range of employees. You will need three controls to collect and display data, a button to save records to the database, a button on click of which you will search a record from the base.
Create a new project. Name it "*People*". Add to it a screen named "*sScreen*". Then add three controls to the screen to input the text, three buttons "*Add*", "*Search*" and "*Refresh*" and three List Box (Fig.44) :

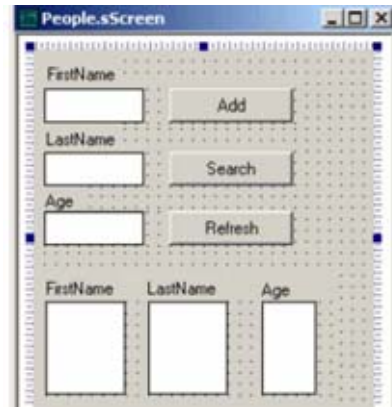


Figure 44 : the position of Controls on the screen

Label the data input and display fields with the help of **Static Controls**. Specify the name of every element. Now that all the visual elements have been added, you have to define the application logic. To start with, create a table where the records will be stored. You can do this by adding a new object –*Database Table*, and then specifying fields, their types and lengths (Figure 45) :

Figure 45

Name this object "*dPeople*". To increase speed, it would be useful to create a *Pocket Access* file, and indicate in the **Provider** field "\Windows\DbPeople.cdb" path. Now, similar to the previous example, add *AddRecord*, *GetRecord* and *Message Actions*. Specify the message box parameters at your discretion. The parameters of the rest of the actions must be as following :

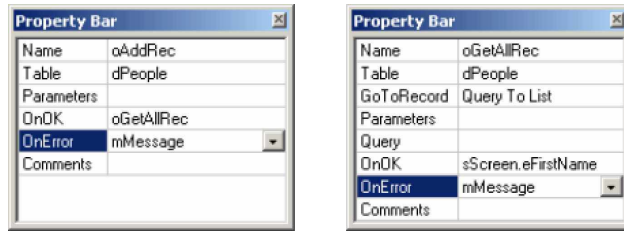


Figure 46 : property Bar for AddRecord and GetRecord Actions

In the AddRecord's **Parameters** field specify controls from which data will be recorded to the database, and in the GetRecord **Parameters** field – controls that will display the table data. The records will be displayed in List Boxes.

Note: *if in the previous examples the names choice were not of great importance, but now you can make sure how important meaning names are, as it is very difficult to remember to what Edit Control names are entered and to what age, and in what List Box they will be displayed.*

Open **Data Link** dialog box and select the necessary control in the corresponding fields, similar to how we did it last time. We have three fields instead of one.

Assignment of Actions to *Add* and *Refresh* buttons
Select *"oAddRec"* in **OnPress** property of *Add* button. Select *"oGetAllRec"* in **OnPress** property of *Refresh* button. This will ensure the record addition to the table when clicking on the *Add* button (the record consists of name, surname and age). As in **OnOk** field of the *oAddRec* Action select *oGetAllRec Action*, the updated database will be displayed on the screen. *Refresh* button is designed to display the whole database.

The new added **GetRecord** action allows searching the recorded entries according to one or several criteria, for example, by surname, name or age. Thus, you set this new created control's property table and name it *"oGetSearchRec"* as in the following figure (Fig.47) :

Figure 47

Open the **Data Link** dialog box and select the same values as for the *oGetAllRec* Action. The next part is to create **Query**. The criteria to form **Record Set** are fixed through the **Query** dialog box (Fig. 48). Compose the demand according to the configuration values of the next figure :

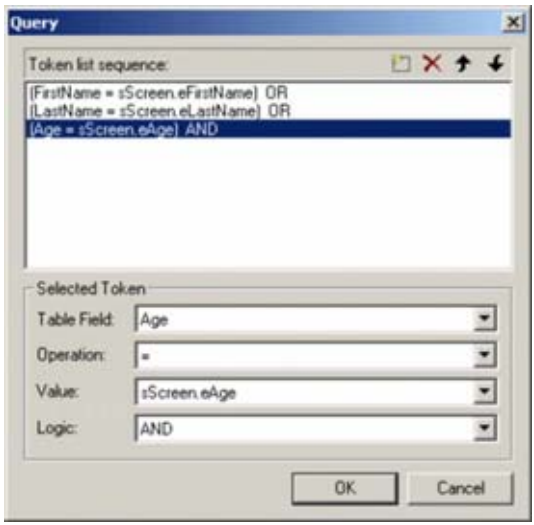


Figure 48

The query is performed according to the following scheme: for every record located in the Table *FirstName* field is compared to the *sScreen.eFirstName* (data input by operator in *firstName* Control). When the correspondence was found, the fields of record would be sent following the special destination in the **Data Link** dialog box.
***Note**: generally, the destinations will be a **List Box** or **Combo Box**. There are no destination with simple Query.*
If the value of surname was not found, then the name would be searched: the records of data of LastName field and sScreen.eLastName control element are compared.
Now you can assign this action to the *Search* button by selecting oGetSearchRec in the **OnPress** field.
If any occurrence was not found, the application gives the message **OnError** in Property table of *oGetRecord* Action.
Assign this action to *Search* button: select *"oGetsearchRec"* for the **OnPress** property of button's property table. Press the **Generate** button and verify if the program is executing on PDA (Figure 49) :

Figure 49 : PDA screen before and after using the search function

Enter some names and memorize them in the base. The purpose of this application was to demonstrate how to display outputs. The search can be done according to several criteria, but the display of output lists is sorted in alphabetic order in every List Box, and this is not desired. In order to avoid such an inconvenience, use *Operation* Action. More details on how to do this will be seen later.
It is clear that a user may need to modify the entered data. An **EditRecord** action will you allow to recall data and edit them.

EditRecord Action

EditRecord action allows you to modify a data record. This action has to be preceded by a **GetRecord** action in which you will select "*First*" for **GoToRecord** property in order to allow the cursor get on the first occurrence corresponding to your research criteria (Figure below).

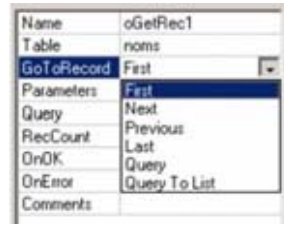


Figure 50

Property	Description
Name	The unique name of the action.
Table	The table name to execute the action on.
Parameters	Defines the fields to edit and the values to assign to them. You'll find more information in the data link paragraph.
OnOK	Defines actions to which the application goes if no errors occur while performing the action.
OnError	Defines an action to which the program goes if an error occurred while performing the action.
Comments	Reserved box for the comments on this action

To create a new **EditRecord** action, do the following steps:

1. Make sure that the project you want to add the action to is active,
2. Click right button on the **Item Bar** and select *Action*. A new action has been added to the project. The properties bar has displayed the action's properties. You can modify the action's name,
3. Select the table you want to edit data on in the **Table** property.
4. Specify links between database fields and controls in which you want to edit your data fields.

Fig. 51

DelRecord Action

DelRecord Action allows to delete one or several records from the database.
Similar to the **EditRecord** action, a **RecordSearch** action has to precede a **DelRecord** action in order to let the program to be posited on the first occurrence of a searched record.

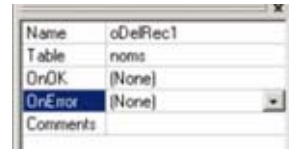


Fig. 52

To create a new **DelRecord** action to the project, do the following steps :

1. Make sure that the project you want to add the action to is active,
2. Right-click on the **Item Bar** and then select in the pop up menu *Insert\DelRecord Action*. As a result, a new action has been added to the active project, and the **Property Bar** displays the new action's properties. You can modify the name of action.
3. In the **Table** property**,** select the table you want to delete data on.
4. Specify links between database fields and controls you want to edit your data fields in.
5. Specify actions to perform in the **OnOK** and **OnError** fields

We've just studied the two actions: EditRecord and DeleteRecord. Let's illustrate the use of them on the next example.
Open the *People* project and add three buttons: *Edit, Del* and *Start View.* Accordingly define their names as *"bEdit", "bDel"* and *"bStartView.* To jump from one record to another, you can create three buttons with *First, Next, Previous* and *Last* using their conventional graphic


and can take advantage of the standard pictures available with the Application Builder. Name these buttons *"bFirst", "bNext", "bPrevious"* and *"bLast*". After all these operations have been done, the screen will look as depicted below (Fig. 53) :

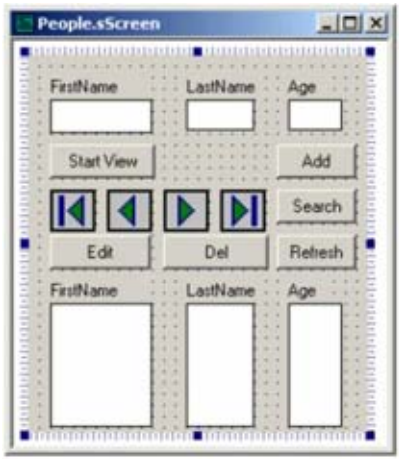


Figure 53 : the screen appearance the stage issued.

Now, in order to make these buttons functional, you have to assign them actions. The last goal is to shift from one record to another, the first one or the last one by pressing the corresponding button. To do this we use **Query** property to select a **Record Set**.

Let's add to our project a **GetRecord** action you name "*GetRec"*. Select from its table of properties a table from which you want data to be retrieved, and then select *"Query"* as the **GoToRecord** value and in **Data Link** dialog box you have to set the following parameters as you see below :

Fig. 54

As a result, this configuration allows to define to what EditControl the record fields will be placed.
Then add four other actions *oGetRecordF, oGetRecordN, oGetRecordP* and *oGetRecordL*. These Actions will help you to surf through the Record Set. Assign them the same parameters as to the *oGetRec* Action with the only difference: in the **GoToRecord** field instead of *"Query"* select *"First", "Next", "Previous"* and "*Last*" respectively.

For the action's back parameter, specify the same screen (by choosing sScreen in the **OnOk** field of every action including *oGetRec*) in case the action is performed without errors, and to create the message if some error did occur (*mMessag*e in the **OnError** field).

Note: Please, pay attention that if the last record in the database is reached and you try to jump to the next, or if you jump from the first record to the previous one, the program will react to such actions as to the wrong ones (**OnError**). Using **Message** action, you inform the operator about the first and last record. Set the buttons properties like on the following Figure :

Name of Action	Property	Value
Start View	OnPress	oGetRec
First	OnPress	oGetRecordF
Next	OnPress	oGetRecordN
Previous	OnPress	oGetRecordP
Last	OnPress	oGetRecordL

Add two functions to edit and delete records. To do this, you add two actions **EditRecord** and **DelRecord** :

Figure 55 : properties of EditRecord & DelRecord Actions

Go back to *Edit* and *Delete* buttons and set them as in the following example :

Name of Button	Property	Value
Edit	OnPress	oEditRec
Del.	OnPress	oDelRec

You have just finished the project's creation. Click **Generate** button  . Having checked accidental errors (if there are some in **Output Bar**, correct the project in accordance with information provided by the output bar), load the People project on the PDA. Start it and add a couple of new records to the database. Click *Refresh* button and you will see the contents of the database sorted in the lists. This will give you an idea of how many records the table contains at that time. You can see that every list is classified in alphabetic mode. It notifies you of the actual number of records in the database.

Press *StartView* button, to dislay the first record in the corresponding **Edit** controls. At the same time the *arrow* buttons are activated. To modify the record content, click the *Edit* button. You can also try to delete some of the records with the help of the *Del* button.

Then click *Refresh* and you will see changes that occured to the database. You can return to the mode display by clicking *Start View*.

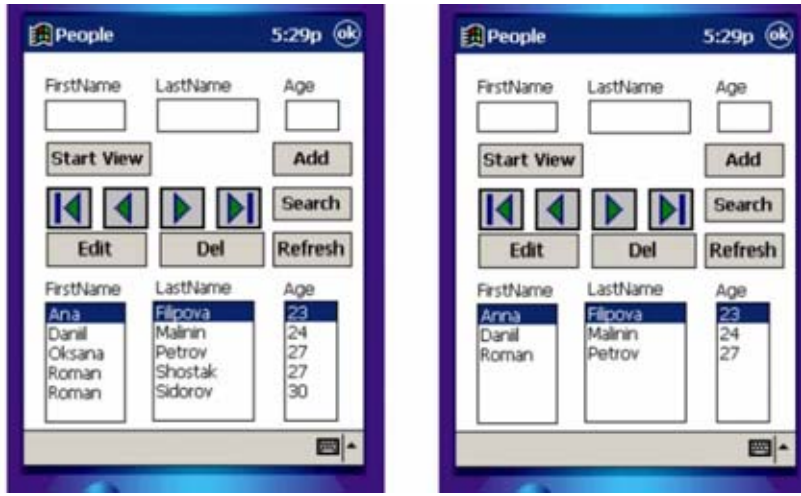


Fig. 56

It often happens that you have to review the whole database. To this effect it would be more convenient to have records consisting of several fields displayed in one unique *List Box* as a single string. You can do this with the help of the **Operation** Action. As a matter of fact, **Operation** Action is a set of sequential simple operations that can be performed when dealing with the contents of **Controls** and **Variables**.

Operation Action

Operation Action performs a sequential set of simple operations like assignment, list clearing, conditional shifts, calculation and so on. This enables you to create program logic. Furthermore, in the event of specific other operation needs, the Application Builder allows you to use functions from external dynamically linked libraries (DLL).

Four properties are defined in the Property Bar. Apart from standard *Name* and *Comments* fields, it also contains **Parameters** and **OnExit** fields. The function of the **OnExit** field is to show where the application has to be linked when all operations that are comprized of this action are performed.

The **Parameters** property defines the sequence of sub-actions and their parameters through the **Operation Action** dialog box.. The picture below shows an example of typical configuration :

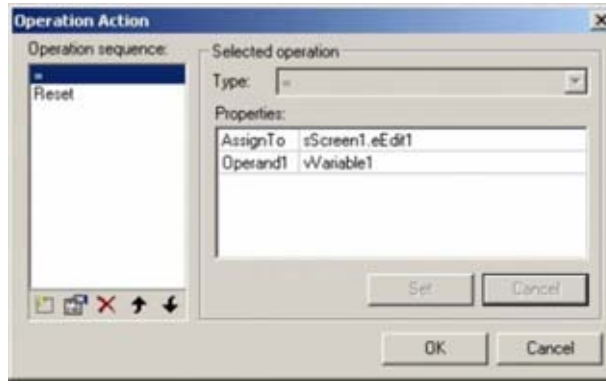


Fig. 58

In this example, an assignment sub-action (=) sets *sScreen1.eEdit1* value to *vVariable1* Variable. After the sub-action is performed, the program will perform a *Reset* sub-action that will clear data list specified in *Operand*.

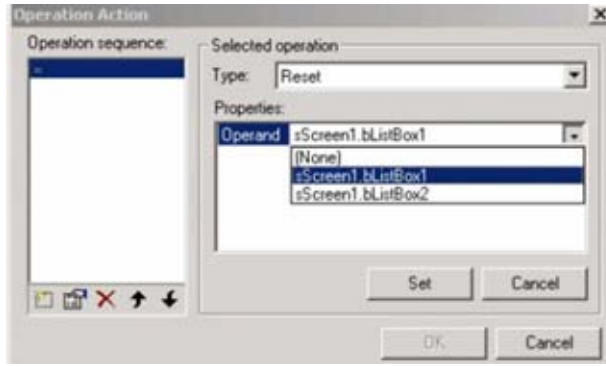


Fig. 59

The different buttons  (*New, Delete, Up*, and *Down*) of this dialog box allow you to constitute your sequence of operations. Edit button allows you to edit the selected action.

Description of the sub-actions :

Name	Description
=	Assigns the **Operand** data item to the **AssignTo** data item
+	Assigns the sum of data items defined in the **Operand1** and **Operand2** properties to the **AssignTo** data item. The sum is appending the **Operand2** string to **Operand1**. Only if the operands are numeric variables the arithmetic sum is performed.

-	Subtracts the **Operand2** Data Item from the **Operand1** Data Item and assigns the result to the **AssignTo** data item. Before operation the operands are converted to the numeric values, then subtraction is performed and the result numeric value is converted to string and assigned to **AssignTo**.
Left$	Extracts the first (leftmost) **Length** characters from the **String** data item and assigns them to the **AssignTo** data item. If **Length** exceeds the string length, then the entire string is extracted. **Left$** is similar to the Basic LEFT$ function.
Right$	Extracts the last (rightmost) **Length** characters from the **String** data item and assigns them to the **AssignTo** data item. If **Length** exceeds the string length, then the entire string is extracted. **Right$** is similar to the Basic RIGHT$ function.
Mid$	Extracts a substring of length **Length** characters from the **String** data item, starting at position Start (1-based), and assigns the substring to the **AssignTo** data item. If there are fewer than **Length** characters in the text (including the character at start), all characters from the start position to the end of the string are assigned. If **Start** is greater than the number of characters in **String**, the **AssignTo** data item is emptied. **Mid$** is similar to the Basic MID$ function.

If () Then,
Else If () Then,
Else,
EndIf
	These sub-actions control conditional branching. They give you the option to run sub-actions depending on run time conditions.
	
	If () Then is a complex statement that may include the **Else**, **Else If() Then** sub- actions, but must be always finished with **EndIf**. The syntax of the operator is : If (expression) Then
	Sub-actions
	[Else If (expression) Then
	Sub-actions]
	[Else
	Sub-actions]
	EndIf
	
	If the logical expression is true, the program runs the following sub-actions until the **Else**, **Else If() Then** or **EndIf** sub-actions. After that, the next sub-action to execute is one followed right away after the **EndIf** sub-action. If the expression is not true, the program looks for and evaluates the expression in the next **Else If ()** Then sub-action. If it is not defined, the application runs sub- actions located between Else and **EndIf**. If the Else sub-action is also not defined, the program executes sub-action defined after **EndIf**.
	You can nest an **If () Then** ... **EndIf** block within another **If () Then** ... **End If** block.
	The program evaluates the expression defined by the **Operand1** and **Operand2** data items with the **Expression** operation.

If (ValidNumber) Then	The **If (ValidNumber) Then** sub-action is similar to **If () Then**, except that expression is calculated differently. The expression is true if the **Number** Data Item is in range between the **Min** and **Max** values.
If (TemplateOK)	The **If (TemplateOK) Then** sub-action is similar to **If () Then**, except that expression is calculated differently. The expression is true if the **String** Data

Then	Item matches to **Template**. Special characters are used to specify the rules for matching: # - Numeric character (0 - 9); % - Alphabetic character (A-Z, a-z); ? - Any character; * - Any character string of any length. For example, template ###-?%% can correspond to the data: 012-AB or 511=CD and the like.
GoTo	Defines action to go.
Reset	Clears contents of list defined in the **Operand** List item
Configure	Specifies set of configuration commands executed at run time.

External function	Executes external function with the **Function Name** located in the **DLL Name** dynamic link library. The DLL has to be located in device Windows directory. The input and output parameters are specified in the **Parameter** property. It is specific sub-action designed for advanced and comprehensive applications.
	The function in the DLL must be defined with C calling convention and has one parameter, for example:
	extern "C" void FuncSample(unsigned short** pData)
	Each parameter of data item parameter array is pointer to zero-terminated set of unicode characters. It is input and output simultaneously. So, be careful with modification of the parameters. It is good practice to return modified value(s) in the globally allocated memory.
	/* This is an example of an exported function.
	The function joins two first parameters and returns the result in last one */
	extern "C" void FuncSample(TCHAR** pData)
	{
	static TCHAR szData[256];
	_tcscpy(szData, pData[0]);
	_tcscat(szData, pData[1]);
	pData[2] = szData;
	}
	
	There is DLL sample located in the installation directory.

SetCtrlState	Disables or enables control selected in the **Operand** list.
SaveDrawing	Save drawing from the control selected in the **Operand** list to the control defined file. To work correctly, the drawing must be visible during execution of the sub-action.
ClearDrawing	Clear drawing from the control selected in the **Operand** list.

To create a new **Operation Action**, you have to do the following :

1. Make sure that the project is active;
2. Click right button on the **Item Bar** and then click in the pop up menu **Insert\Operation Action**.
3. Change the by-default name.
4. Assemble the sub-actions in **Operation Action** dialog box.
5. Determine the value of **OnExit** property to specify to what Action goes the application.

An inconvenience in our *People* example was that the records fields in every Control were sorted on alphabetic order and it was impossible to determine which surname is corresponding to a given name and so on. **Operation** actions and **Variables** in our project provide you with a more effective solution.
Using the «+» action and adding separators, we can combine all three records fields into one line and deliver the whole database, line by line, into a List Box.

To do this we will add the following elements to our project :

1. Two buttons, *Search* and *Refresh*, which will perform search and reset actions
2. Add to the project the following **GetRecord** actions :
   oGetRecNN – action with *Query* parameter in order to form **Record Set**
   oGetNextN – switch to the next record
3. Add to the project the following operations:
   oResetList – operation to delete the list contents oAddToList – operation to add a record to the list

Operation algorithm looks like this (figure 60) :

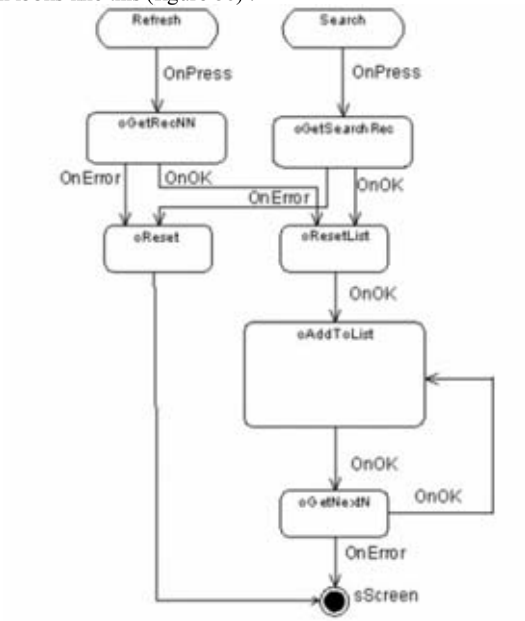


Figure 60 : program operation algorithm

Comments :
Every button performs a corresponding action (constitution of a **Record Set** or a Search). If OK, the application starts by deleting the contents of a list, then, successively :
The first record of **Record Set** is put in the list,
The next record is taken,

If OK, this record is added to the List,
The cycle is repeated till the end, unless an error occurs.

To perform the functions of the given algorithm, you have to :
Open the *People* project. Delete two *List Boxes* and rename the third one to *"bListAll"*.
Parameters of all actions that derived data to the deleted List Boxes should be reassigned.
Before doing so :

1. Add three Variables *vFirstN, vLastN, vAge* to the project that will help to stock the extracted information from the Base,
2. Define their type as *String*,
3. Also we need some separators that will be stored in variables *vKav* and *vDoublKav*. In the **Data** field put the sign «'» for the first variable and «' '» for the second. We will use these variables for joining lines when separating fields.
4. Add two **GetRecord** actions to the project with the following parameters (Fig. 61) :

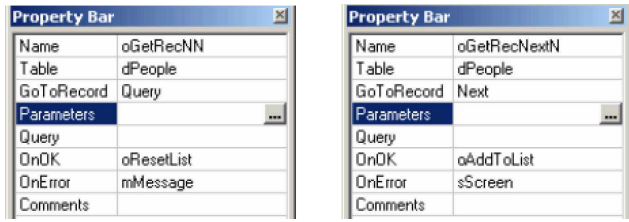


Fig. 61

In the **Parameters** field of these actions define the variables' names, to which the records fields value will be entered. (Fig. 62)

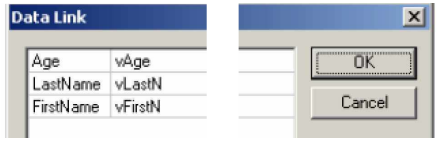


Fig. 62

Since we want to extract the contents of the whole database, the **Query** field is left empty for these actions. To delete data in the list, let us add **Operation** action to the project. Name this Action *oResetLis*t and in its **Parameters** property open the **Operation Action** dialog box. In the **Type** field select the *Reset* action. Define the name of the Control *sScreen.bListAll* in the **Operand** field. We need another operation to delete the contend of the list for the case when the database is empty. After performing **GetRecord** action, the program will treat the empty database as an error in **OnError**. Name this Action *oReset* and set exactly the same parameters as for the *oResetList* action, except the **OnExit** field that will have as value an **Edit Box**.
Now let us create an action which will concatenate the three fields into one line and put it into the *List Box*. Add a new **Operation** action named *oAddToList* to the project. In the **Operation action** dialog box choose the «+» action and define the following *Operands*' values (Figure 63) :

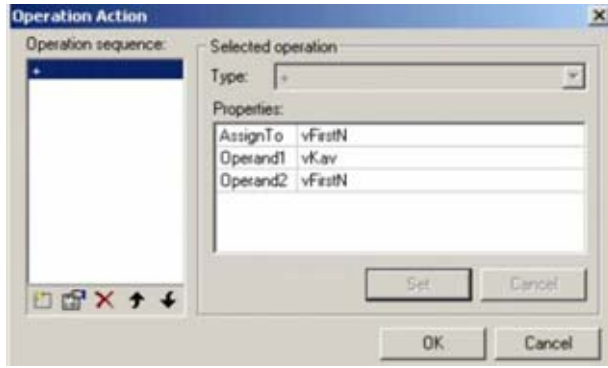


Fig. 63

Press the **New** button and add a similar action to concatenate : *vFirstN* and *vDoublKav* in *vFirstN*. Then go on in order to get : vFirstN= vKav + vFirstN + vDoublKav + vLastN + vDoublKav + vAge + vKav.
Now, we have to assign the final Value *vFirstN* to the *sScreen.bListAll* list (Fig. 64) :

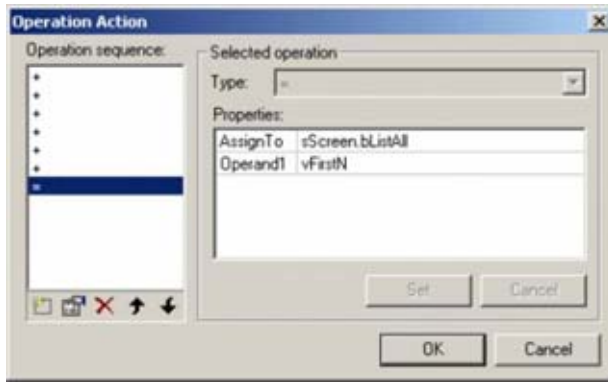


Fig. 64

Now, let us think of the presentation of the data into the list, according to the selection criteria. It is necessary to use the *oGetSearchRec* Action to select elements. In this case, the records found will not be sent to a list, but assigned to variables. To do that, define a corresponding variable for each field in the **Data Link** dialog box of this action.
We have created all the necessary actions and operations; now we must indicate links between them and assign their execution to corresponding buttons.
Go back to the properties of the following actions and buttons :

- *Refresh* button : **OnPress** : *"oGetRecNN"* ,
- oGetRecNN Action : **OnOk** : "oResetList", and **OnError** : "oReset".
- oResetList Action : **OnExit** : "oAddToList"
- oAddToList Action : **OnOk** : "oGetRecNextN"
- OGetRecNextN Action : **OnOk** : "oAddToList", and **OnError** : "sScreen"

- *Search* Button : **OnPress** : “oGetSearchRec”
- OGetSearchRec : **OnError** : “oReset” (End of List error message), and **OnOk** : “oResetList”
- OResetList Action : properties have already been defined

Click the **Generate** button to generate the application and load it on PDA.
In your PDA, open the application, add some new records and press the *Refresh* button. Select a name and perform the search (Fig. 65) :

Fig. 65

If anything goes wrong, go back to the project and check whether or not you have correctly defined actions and operations sequence and their parameters.

Let us slightly expand the functionality of the application by adding new actions to find maximal and minimal ages of company employees.
Define two Variables *vMax* and *vMin*. Assign the *vMax* Variable some low number, so that it cannot be encountered in the table, and the vMin Variable – some very high number. We will compare the Age field value with these variables’ values. If we encounter a number which is smaller than *Vmin* value, the variable will be assigned to this value. The same goes for Vmax, which will be assigned a value that is greater than the value already stored in it. Let us add to our project oGetRec_New Action with the parameters specified on the Fig. 58. The result of this action will be the assignment of the value of the field corresponding to the first record of table to *vAge* Variable
(precisely, all fields’ values are assigned to corresponding variables; however, we use only value

Fig. 66 : oGetRec_New action parameters

Add another Screen on which we can place fields for minimal and maximal values and a button for switching back to *sScreen* (Fig. 67).

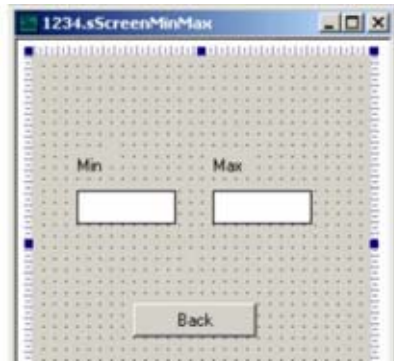


Fig. 67

Now let us create an operation, which will compare variables' values with the field value. Call it oExecute. In its **Operation Action** dialog box specify the following sub-actions :

if () then

operands' values will be the following :

Operand1 : vMax
Expression : <
Operand2 : vAge
=
AssignTo : vMax
Operand1 : vAge

End if

if () then

Operand1 : vMin
Expression : >
Operand2 : vAge
=
AssignTo : vMin
Operand1 : vAge

End if
As a result, the dialog box will look like this (Fig. 68) :

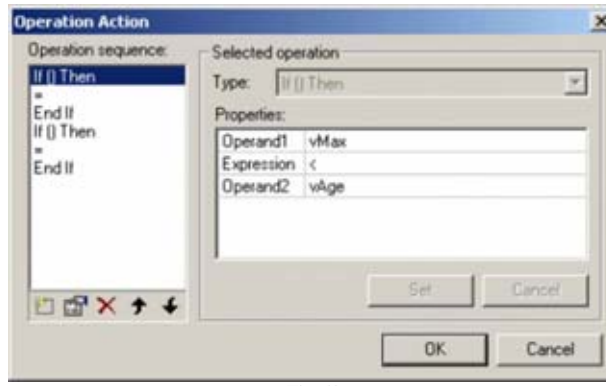


Fig. 68

In order to pass to a new record and assign his fields' values to variables, it is necessary to add *oGetRecNextNew* Action. In its properties table specify value "*Next"* in the **GoToRecord** field, while in the **Parameters** field specify the variable to which the values of the corresponding fields will be assigned (similar to *oGetRec_New* Action). Then you must add another operation, *oPrint*, which will assign final values of *vMin* and *vMax* Variables to fields to display on screen and determine the sequence of **Actions** and **Operations**.
Add to the main screen *sScreen* a Button named *Min/Max*, and parameter **OnPress** property for oGetRecNextNew. By clicking this button, operations will be executed in accordance with following algorithm (Fig. 69) :

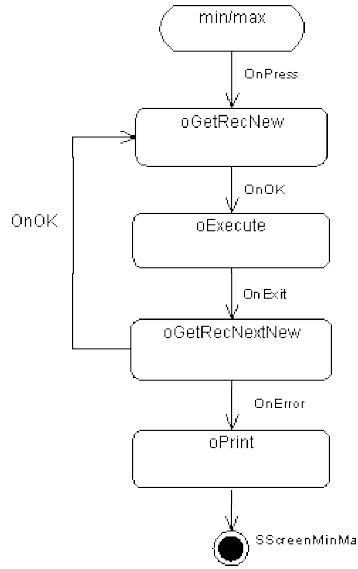


Fig. 69 : Algorithm

The program successively retrieves one record at a time from the database. When there are no more records in the database, an error ocurs and the program outputs by **OnError**. Variables' values, which store minimal and maximal ages, are displayed on the screen.
After setting of the sequence of all actions and operations you can transfer the program to your PDA and test it. Add some new records to the database, just enough to easily determine minimal and maximal ages of all employees. Press the *Min/Max* button and check whether displayed values agree with those you have determined by yourself. Go back to the main screen and repeat the procedure after having added or removed some records.

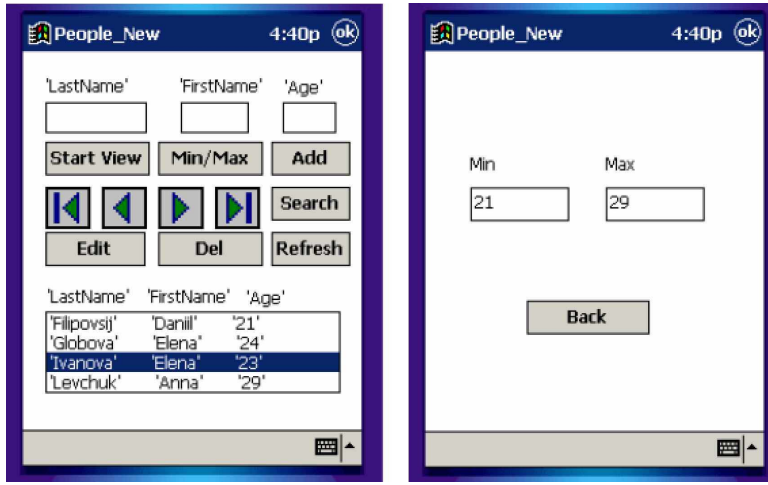


Fig. 70 : result of the search of minimal and maximal values

So far we have been using only string and numeric variables, but as mentioned in the variables section, there is another variable type - *Date&&Time*. In order to understand the purpose of this kind of variable, let us carry out a small example task. We will create an application, which will have one screen with input field, a button and a text field. When the button is pressed time of the moment when it was pressed will be displayed in a specified format (depending on the parameters specified in the text field).

Create a new project and name it *Data&Time*. Add a screen named *sScreen*, an **Edit Control** named *eDataTime* and two **Static Controls**, one named *cDateTime* – for information output and the other one named *cInfo* for informational purposes.

Add **Frame Rect** to the project and place it in such way that the *cDateTime* **Static Control** would be inside it. (Fig. 71)

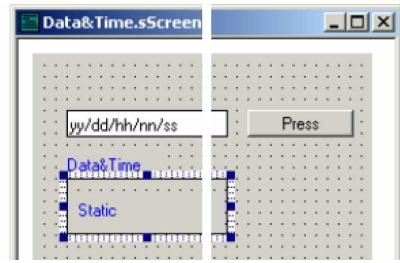


Fig. 71

Remove the text "value of data's property "from the *cData* field of the **Static Control**, and for the other *cInfo* type in that field *Data&Time*. You may also change the color of these controls, e.g. choose Blue. In **Data** property of **Edit Control** change to *yy/dd/hh/nn/ss*. Add a Variable to the project and name it *vDataTime.* Specify its type - *Date && Time*. Add an *Operation Action* named *oOperation* with the following parameters :

Operation : =
AssignTo : **vDataTime**
Operand1 : *sScreen.eDataTime*
Operation : =
AssignTo : **sScreen.cDateTime**
Operand1 : *vDataTime*

In the **OnExit** field choose *sScreen.eDataTime*. Now go back to the button and name it *bButton*, having specified *Press* in the **Button** field, while in the **OnPress** field choose *oOperation*. Go to project properties and in the **OnStart** field set *sScreen.eDataTime*. In the **Platform** field choose the device type and click **Generate**.
Open the *Data&Time* application on your PDA and click the *Press* button. Time and date will be displayed in the following format. Click it again. The seconds' value has changed (the last number). Delete data from the input field and type *"t"* in it. Time will be displayed in standard format

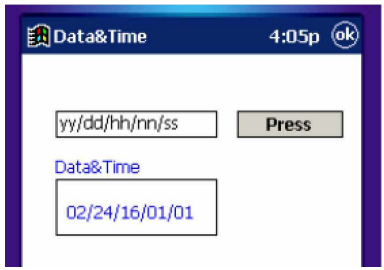


Fig. 72

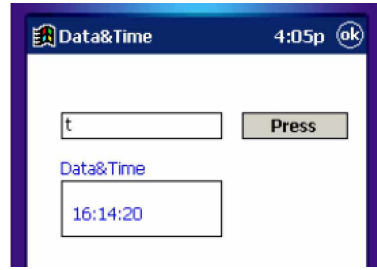


Fig. 73

We get different date/time format when pressing the button, depending on symbols we have specified in the input field.

CopyTable Action

CopyTable action gives us a possibility to exchange data between readers and the host computer. The result of this action is that all records are copied from the specified database table to the host Microsoft access database or from the database on the host computer to PDA. If the specified table exists, the method of performing this action depends on the **Overwrite** property. In order to work with this action it is necessary to launch the data synchronization application on the host computer. Also there must be a connection established with PDA using **ActiveSync** software.
In the properties table there are several fields:

Name	The name of action
Device Table	A specified table located on PDA, which is involved in data exchange
DesktopFile	a database located on the host computer
Direction	In this field you may choose the data transfer direction: PDA to host computer or host computer to PDA.
Overwrite	If the target table already exists, you may choose Yes in the Overwrite field to overwrite the existing table. If you choose No, there will appear an error in the

	case when the target table exists.
OnOK	In the **OnOK** field the action that will be executed in the case of successful copying completion is specified.
OnError	In the **OnError** field the action that will be executed in the case of an error is specified.

To create a new **CopyTable** action, do the following steps :

1. Make sure that your project is active;
2. Right click on **Item Bar** and choose from pop up menu *path Insert\CopyTable Action*. The new action is appended to the active project and the Properties pane shows the properties of the action.
3. You may change the name of the action
4. Define destination and source tables in the **Device Table** or **DesktopFile** entries.
5. Specify the direction of data transfer in the **Direction** property.
6. Check the **Overwrite** property value.
7. Specify actions on the **OnOK** and **OnError** events.

Let us go back to our *People_New* project and implement data exchange between host computer and PDA in it.
Open the project and add two buttons *Desktop2Device* and *Device2Desktop* to it in the following way (Fig. 74) :

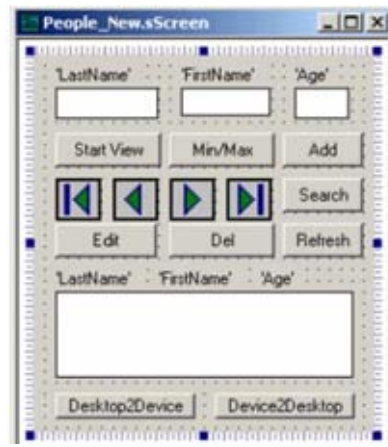


Fig. 74

and two **CopyTable** actions with parameters as shown in the Fig. 75 :

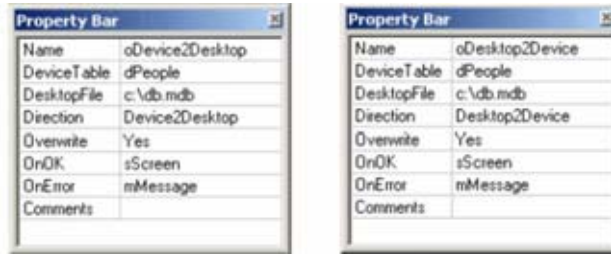


Fig. 75

Assign *oDesktop2* Action to the *Desktop2Device* button and *oDevice2Desktop* Action to the *Device2Desktop* button.
Click **Generate** and launch the program on your PDA. On the host computer click **Start\Programs\IDEAM\IDEAM CE\IDEAM Sync.** Data will not be transferred without this. Go back to your PDA. Add some records to the database and click *Refresh* (Fig. 76) :

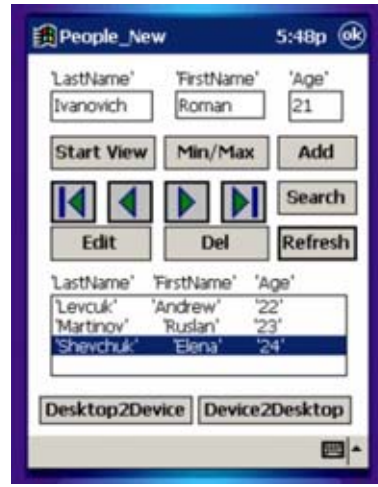


Fig. 76

Now click the *Device2Desktop* button. After data transfer completion open the file c:\db.mdb using Microsoft Access on the host computer and change some records in it. Save the file and click the *Desktop2Device* button on your PDA. After the completion of data transfer from the host computer click *Refresh* button on your PDA. Now you get on the screen the modified data (Fig. 76) :

Calculator : an example of using Operation Actions

Let us proceed to an example illustrating the use of different sub-actions (+,-,*,/). This example shows a simple calculator, which allows you to add and subtract numbers as well as perform some actions on strings.

So, proceeding to a new project creation. Let us try to make our calculator look and work as close as possible to a standard one.
Analyze the operation principles of a domestic calculator.

1. The first thing we do is input data (operand 1) in a field. Then after clicking on a button the data we have entered is stored, as well as the operation corresponding to the button.
2. Then, another operand is entered.
3. The relevant operation is performed by clicking on a button. The result is stored and so on.

Since we want to work both with digits and strings, e.g. input them in the same field, considering the Application Builder's features, it will be simpler to create two identical screens, one of them with a field to enter text and buttons to work with strings, and another screen with a field to enter digits, with no buttons oriented to string operations and with buttons for operations with digits. Thus the user will think that he works with the same screen but with buttons becoming active or inactive depending on the data type.
First add one screen named *sStrings*. Place the elements on it in the following way :

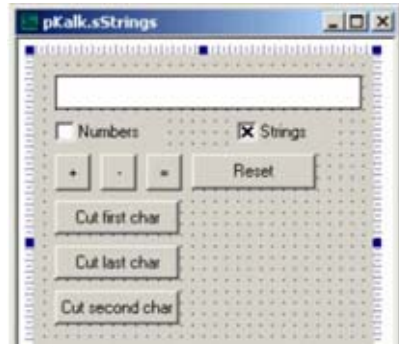


Fig. 77 : Controls placement on the screen

In Check Boxes the user will select the operation mode; the «+» button will execute strings concatenation or numbers addition; «-» will be used only to subtract numbers and on the screen for strings mode it will be permanently inactive; «Cut first char», «Cut last char», «Cut second char» buttons are aimed at operations with strings only to allow the user to extract the first symbol of the string, the last one, and the second from the beginning of the string respectively; «=» button, like in domestic calculators, serves for result output; «Reset » is to clear the input field.

Create a new screen *sNumbers* and copy all the common elements from the *sStrings* screen to it. Placement of the buttons must be absolutely identical, to avoid common fields to switch when the user will switch from one mode to the other.

Define the following variables in the project (in the brackets parameters are specified: variable type and default value):
vStringResult (String, “”) : Variable to store results of operations with strings.
vNumResult (Numeric, 0) : Variable to store results of operations with numbers.

Variables to store operations’ codes and their default values :

- **vStringSum (Numeric, 1)** – strings addition operation code.
- **vStringLeft (Numeric, 2)** – string's first symbol extraction code.
- **vStringRight (Numeric, 3)** – string's last symbol extraction code.
- **vStringMid (Numeric, 4)** – string's second symbol extraction code. Let us set the quantity of symbols extracted equal to one.
- **vNoOperation (Numeric, 0)** – code designated to signify absence of stored operations when starting the calculator application, changing between strings and numbers, or during output operation when «=» is pressed.
- **vNumSum (Numeric, 5)** – numbers sum operation code
- **vNumSub (Numeric, 6)** – numbers subtraction operation code

Variables listed above are set by default and do not change during program operation (constants).
Condition variables:

- **vNextStep (Numeric, 0)** – variable to store pressed button operation code.
- **vCurrStep (Numeric, 0)** – variable to store previous pressed button operation code.
- **vEmptyString (String,"")** – auxiliary variable to empty strings.

So as not to get confused with data types, let us perform separate calculation operations for strings and numbers.

When the application is launched, *vCurrStep* and *vNextStep* Variables are equal to zero. Each button has a corresponding action (specified in the **OnPress** field), which stores a corresponding code to the *vNextStep* Variable, followed by switching to *oExecuteCalc* Action. Depending on *vCurrStep* Variable's value, a corresponding sub-action is executed. When the sub-action is completed, *vCurrStep* Variable is assigned the vNextStep Variable's value.

Create the following sub-actions and arrange them in accordance with the buttons:
Specify for the «+» button in the **OnPress** field Action oStringSum with the following parameters:

1. =
AssignTo: *vNextStep*
Operand1: *vStringSum*

For the «=» button in the **OnPress** field specify oStringEqual Action with the following parameters:
1. =
AssignTo: *vNextStep*
Operand1: *vNoOperation*

For the «Cut first char» button in the **OnPress** field specify oStringLeft Action with the following parameters:
1. =
AssignTo: *vNextStep*
Operand1: *vStringLeft*

For the «Cut last char» button in the **OnPress** field specify oStringRight Action with the following parameters:
1. =
AssignTo: *vNextStep*
Operand1: *vStringRight*

For the «Cut second char» button in the **OnPress** field specify oStringMid Action with the following parameters:

1. =
AssignTo: *vNextStep*
Operand1: *vStringMid*

The **OnExit** field for all actions listed above is set to *oExecuteCalc*.
Add an **Operation** action to the project and name it *oExecuteCalc*. This sub-action is aimed at operation with strings. In the **Parameters** dialog box create a sub-action sequence **If() Then ... Else If() … End If** in the following way :

Sub-action
If() Then
Operand1: vCurStep
Expression: =
Operand2: vStringSum
+
AssignTo: vStringResult
Operand1: vStringResult
Operand2: sStrings.eEdit
Else If() Then
Operand1: vCurStep
Expression: =
Operand2: vNoOperation
=
AssignTo: vStringResult
Operand1: sStrings.eEdit
Else If() Then
Operand1: vCurStep
Expression: =
Operand2: vStringLeft
Left$
AssignTo: vStringResult
String: vStringResult
Length: 1
Else If() Then
Operand1: vCurStep
Expression: =
Operand2: vStringRight
Right$
AssignTo: vStringResult
String: vStringResult
Length: 1
Else If() Then
Operand1: vCurStep
Expression: =
Operand2: vStringMid
Mid$
AssignTo: vStringResult
String: vStringResult
Start: 2

Length: 1
End If
=
AssignTo: vCurStep
Operand1: vNextStep
If() Then
Operand1: vCurStep
Expression: =
Operand2: vNoOperation
=
AssignTo: sStrings.eEdit
Operand1: vStringResult
Else
=
AssignTo: sStrings.eEdit
Operand1: vEmptyString
End If

Set the **OnExit** field of this action to *sStrings.eEdit*
operation algorithm when operating with strings looks like this (Fig. 78) :

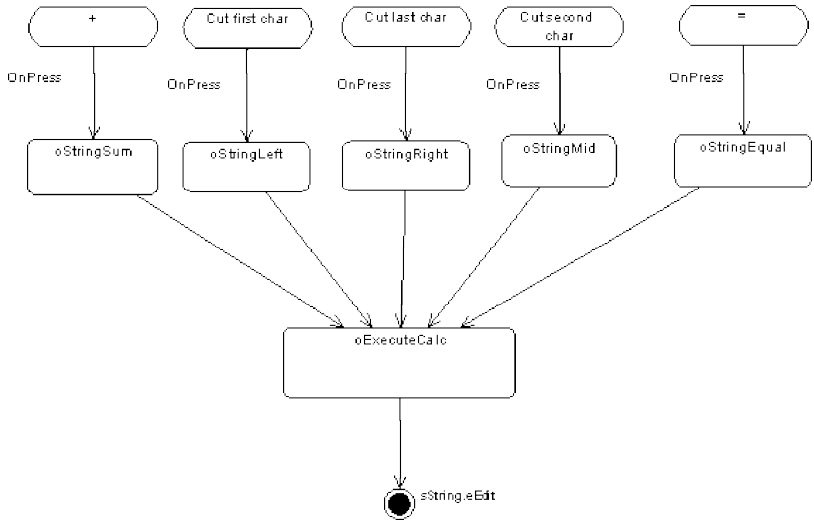


Fig. 78

Let us consider more closely one algorithm branch, as other branches work in a similar way. Suppose you have entered a string in the input field and pressed «+». Button's **OnPress** event is associated with *oStringSum* **Operation** action, which assigns the variable *vNextStep* the value of the *vStringSum* constant. After that the oExecuteCalc operation is executed.
The result is stored in the *vStringResult* Variable. When the operation is completed, the *vCurrStep* Variable is assigned the value of vNextStep variable. Then if vCurrStep =

vNoOperation, the value of *vStringResult* is entered in the input field, otherwise the input field is cleaned, i.e. it is assigned to the value of *vEmptyString* variable.

When operating with numbers, calculations are performed using a similar algorithm. Switching between screens is performed with the help of **Check Boxes**. Only one **Check Box** can be selected at a time. It is necessary to add to the project two **Operation** actions, which will select or deselect **Check Boxes**. Name them *oGoToNumbers* and *oGoToStrings*. Parameters for them will include :

In the **Parameters** dialog box define a sequence of sub-actions for *oGoToNumbers* Action :

1. =
AssignTo: sStrings.chbNumbers
Operand1: vFalse
2. =
AssignTo: vStringResult
Operand1: vEmptyString
3. =
AssignTo: vNumResult
Operand1: vFalse
4. =
AssignTo: vNextStep
Operand1: vNoOperation
5. =
AssignTo: vCurStep
Operand1: vNoOperation
6. =
AssignTo: sStrings.eEdit
Operand1: vEmptyString
7. =
AssignTo: sNumbers.eEdit
Operand1: vEmptyString

The result of this sequence will mean all variables will be cleared and the **Check Box** will be set

For *oGoToStrings* Action in the **Parameters** dialog box define a similar sequence of sub-actions :

1. =
AssignTo: *sNumbers.chbStrings*
Operand1: *vFalse*
2. =
AssignTo: *vStringResult*
Operand1: *vEmptyString*
3. =
AssignTo: *vNumResult*
Operand1: *vFalse*
4. =
AssignTo: *vNextStep*
Operand1: *vNoOperation*

5. =
AssignTo: *vCurStep*
Operand1: *vNoOperation*
6. =
AssignTo: *sStrings.eEdit*
Operand1: *vEmptyString*
7. =
AssignTo: *sNumbers.eEdit*
Operand1: *vEmptyString*

Set the **OnExit** field of these two sub-actions to *sStrings.*
Select on the *sStrings* screen the **Check Box** named *chbNumbers* and in its **GoOnDataChange** property select the o*GoToNumbers* action. For the **Check Box** named *chbStrings* leave this field empty. On the *sNumbers* screen select the **Check Box** named *chbStrings* and in its **GoOnDataChange** property select the *oGoToStrings* Action, while leaving the field of the *chbNumbers* **Check Box** empty.
In order to switch to the numbers screen, one has to select the *chbNumbers* **Check Box** on the strings screen. The **GoOnDataChange** event of the **Check Box** calls the *oGoToNumbers* action, which sets the *chbNumbers* **Check Box** on the *sStrings* screen unselected, as well as clears the *vNumResult, vStringResult* Variables and the input fields, while assigning state variables the value of *vNoOperation* constant.

When the action is completed, the switch to the *sNumbers* screen is performed. Switch to the strings screen is performed in a similar way using the *oGoToStrings* action.
For the *Reset* button, which is located on *sStrings* screen, in the **OnPress** field set *oGoToStrings* action.
Operation with numbers is performed using the same algorithm as with strings.
When switching to the numbers screen, values of *vCurrStep* and *vNextStep* are set to 0. After pressing a button a corresponding sub-action is called :
The «+» button on event **OnPress** calls the *oNumSum* Action, which has the following specified in its **Parameters** field :
1. =
AssignTo: *vNextStep*
Operand1: *vNumSum*

Button «=» select *oNumEqual* for **OnPress** property, for **Parameter** property of the operation, enter :
1. =
AssignTo: *vNextStep*
Operand1: *vNoOperation*

Button «-» select *oNumSub* for **OnPress** property, for **Parameter** property of the operation, enter :
1. =
AssignTo: *vNextStep*
Operand1: *vNumSub*

The **OnExit** property of Actions listed above is set to *oCalcNum*.

Add an **Operation** Action to the project (similar to *oExecuteCalc*) and name it *oCalcNum.* In the **Parameters** dialog box create a sequence of sub-actions **If() Then ... Else If() … End If** in the following way :

1. Sub-action **If() Then**
Operand1: vCurStep
Expression: =
Operand2: vNumSum
2. +
AssignTo: vNumResult
Operand1: vNumResult
Operand2: sNumbers.eEdit
3. **Else If() Then**
Operand1: vCurStep
Expression: =
Operand2: vNoOperation
4. =
AssignTo: vNumResult
Operand1: sNumbers.eEdit
5. **Else If() Then**
Operand1: vCurStep
Expression: =
Operand2: vNumSub
6. **-**
AssignTo: vNumResult
Operand1: vNumResult
Operand2: sNumbers.eEdit
End If
7. =
AssignTo: vCurStep
Operand1: vNextStep
8. **If() Then**
Operand1: vCurStep
Expression: =
Operand2: vNoOperation
9. =
AssignTo: sNumbers.eEdit
Operand1: vNumResult
Else
10. **=**
AssignTo: sNumbers.eEdit
Operand1: vEmptyString
11. **End If**
12.

In the **OnExit** field of this action set *sNumbers.eEdit.*
On the numbers screen, the buttons "+" and "-" are active as are the call *oNumSum* and *oNumSub* actions, which assign the *vNextStep* variable and the values of *vNumSum* and *vNumSub* respectively. After completing these actions the *oNumCalc* Action is executed, which is similar to *oExecuteCalc* action as it executes sum or subtraction operation and assigns *vCurrStep=vNextStep*, followed by clearing the input field.

Fig. 79 : The finished application executed on PDA

Database queries : SQL Action

SQL Action allows to create and execute SQL queries. This function gives more options for the program. The application supports the SQL language queries in accordance with ADOCE technology. Comprehensive information on queries and SQL language can be found in SQL Reference for Windows CE or Microsoft ADOCE user's guide.
The SQL actions possesses the following properties :

Name	The name of action
Table	The name of table on PDA the query will be executed on
SQL	Description of SQL query
OnOK and OnError	Idem for other actions

SQL Operators example in the SQL property :

DELETE FROM DBTABLE	Delete the records from DBTABLE
DROP TABLE DB1	Delete the records from DB1 Table
CREATE DATABASE ' \Windows\Start Menu\db.cdb '	Create Pocket Access database for Windows CE
DROP DATABASE ' \Windows\Start Menu\db.cdb'	Delete the base from PDA

To create a new **SQL Action**, carry out the following steps :

1. Make sure that your project is active;
2. Right-click on **Item Bar** and choose from pop up menu *Insert\SQL Action*. The new action is appended to the active project and the **Property Bar** shows the properties of the action.
3. Change the name of the action
4. Specify the table you want to execute the SQL query on in **Table** property.
5. Specify actions on the **OnOK** and **OnError** events.

Remote Database Queries & Call towards Remote Applications : Transaction Action

If you want the application to execute queries towards a database situated on another platform than the PDA, or to trigger a logical process running on a remote platform, you have to use the Communication Server (not installed with the product). **Transaction Action** allows to pass commands and queries to a Communication Server.

How to execute the transaction :

- The Communication Server has to be installed on the Host computer.
- The specified communication parameters in **Configuration** properties have to correspond to the Communication Server properties. (By default, the parameters are identical in the Communication Server and the Application Builder software).
- The module receiving data has to be active and well configured.
- The execution of *Transaction sub-action* has to be carried out by message of module in space of time named Time out…

Properties of Transaction Action :

Property	Description
Name	The unique name of the action
SendData	Command to send to the Communication Server
RecvData	Elements, to which data will be assigned. If the value is None, the program will not recover any data
FRSH Format	Determine if received data is in FRSH format. If the value property is on "Yes", the application is waiting for received data according to the FRSH format. The data will have the next syntax : The first record is : CCCCFFF<CR> Where : CCCC : 4 numbers indicates the number of records FFF : 3 numbers for the number of fields The next records have the format (for M fields and N records): LL1LL2 . . . LLMdata11data12data13 ... data1M<CR> LL1LL2 . . . LLMdata21data22data23 ... data2M<CR> LL1LL2 . . . LLMdata31data32data33 ... data3M<CR ... LL1LL2 . . . LLMdataN1dataN2dataN3 ... dataNM<CR><SOH> where : LL1 – 3 numbers for the length of first data field (data11) LL2 – 3 numbers for the length of second data field (data12) and so on. The format is used in the database module of the Communication Server.
RecvFRSH	Assign received data to elements in *Received Data Dialog* dialog box. This property is valid in the case if the **FRSH Format** value is *True* value
OnOK	Property that determines the performing action if the execution succeed
OnError	Property that determines the performing action if the an error has occurred

Comments	Comments on Action

In **SendData** of **Transaction Action** property the transaction that will be passed to the the Communication Server is created. This transaction is created with tokens and assembled from **Server Command** dialog box. (Figure 80) :

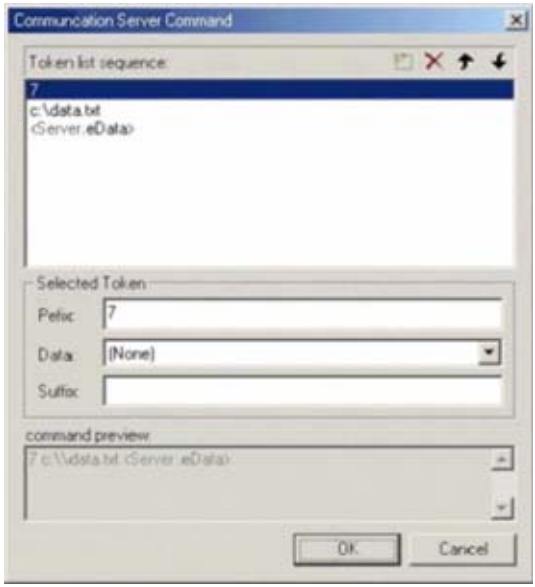


Fig. 80

This figure presents a command format designed for **PwFoxPro** module of the Communication Server. For this module you'll find module's number in **Prefix** zone 7 number; "c:\file.txt" represents the name and path to which data will be sent. The **Data** field contains the control or variable from which the data stems. To learn the commands' syntax, go to the technical documents of the Communication Server software.

RecvData property serves to specify the control that will received data back from Communication Server module. If this property is on *None,* no information will be sent back to application from the module.

FRSH Format property is used for modules that send back data in FRSH format. For modules that use this format, the data flow will be as follows:

the command defined in **SendData** property is sent to the Communication Server. If **FRSH Format** is "Yes", then the Communication Server will send back data according to the following data flow:

1. The Communication Server sends a first message giving the number of records and number of fields,
2. The application sends the next request for reception composed of two tokens, the first is the module's number and the second is *$NextRec$.*

Data is assigned in **RecvFRSH** property (Sample figure 81):

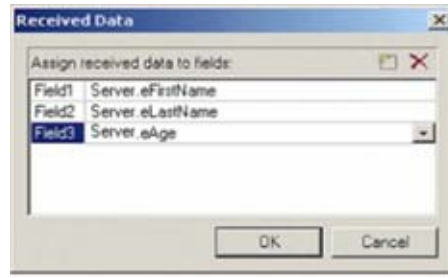


Fig. 81

To create a new **Transaction Action**, carry out the following steps :

1. Make sure that your project is active;
2. Right-click on **Item Bar** and choose from pop up menu *Insert\Transaction Action*. The new action is appended to the active project and the Property Bar shows the properties of the action.
3. Compose a command in **SendData** property.
4. If return of data, specify in **RecvData** and **RecvFRSH** property to which Variables they will be assigned.
5. Specify actions on the **OnOK** and **OnError** events.

To understand data treatment in **Transaction Action**, we will use an example.

Create a new project and name it : *Transaction.* Add a screen and name it aerver and place controls in it (Fig. 82) :

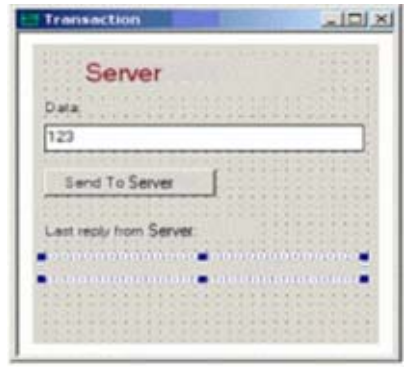


Fig. 82

Name an empty **Static Control** as *cReply.* We will use it to receive data from the Communication Server. Name **Edit Control** *eData* that serves to pass data to the Communication Server.

Add two message windows in project and parameter the properties as in the next figure (figure 83 a, b) :

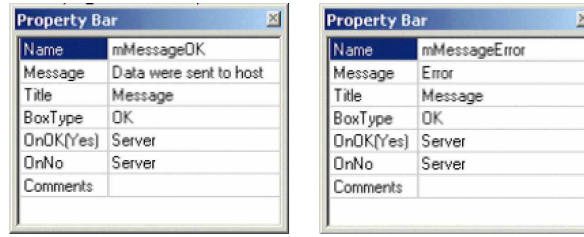


Fig. 83 a : parameters of *OnOk* and *OnError* Message windows

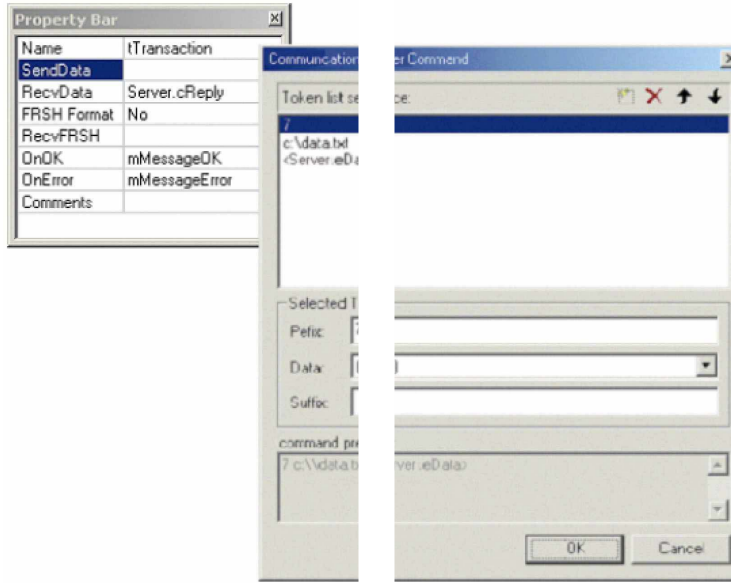


Fig. 83 b : Transaction Action setting

Assign to *Send To Server* button the execution of **Transaction Action** and then download the project on PDA. Start the Communication Server on Host computer (**FoxPro** module has to be active and set as in Figure 84) :

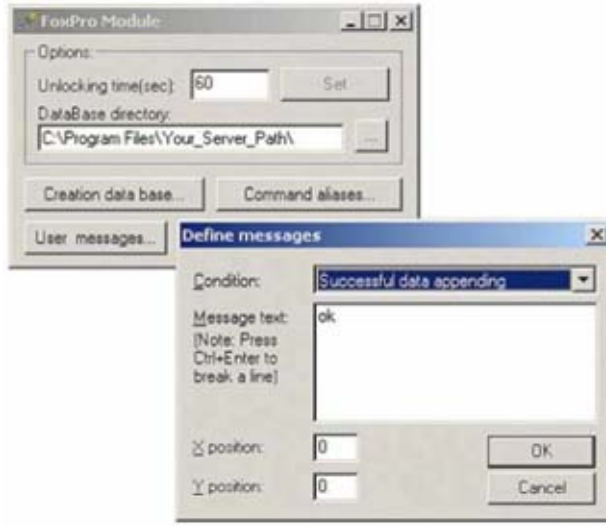


Fig. 84

While the Communication Server is running, go back to PDA and enter data in input field, then click *Send To Server* button. If the transaction is correctly executed, the figure below will be displayed on your PDA (texts depend on entered information and the application's reply) Figure 85 :

Fig. 85

If an error occurred during the transaction, the Communication Server will return a message with an error code.

Ftp Action

Ftp Action allows you to send and receive files with FTP protocol. To correctly execute this Action, configure PDA for reception and transmission with TCP/IP protocol. If while executing

of action, the user pressed Cancel button in Download (Upload) field, the program would finish the FTP transfer and connect the special action in **OnError** property.

Fig. 86

This action has these properties:

Name	Name of action
Direction	Indicates a transfer direction (from host computer to PDA => download, from PDA to Host => upload
LocalPath	Indicates the directory path on PDA
RemotePath	The server's name FTP and the directory path
FileMask characters,	The mask specifies the files copied. This mask uses standards "*" and "?". In user and password properties indicates the name that
	user goes in delete server with.
User	Informs if the directory is protected
Password	Informs if the directory is protected
DeleteSrc	Indicates if the files will be deleted after being copied
OnOK OnError	Indicates an executing action after the copy is succeesful Indicates an executing action after an error has occurred or the

To create an **Ftp Action**, carry out the following steps :

1. Make sure that your project is active;
2. Right-click on **Item Bar** and select from pop up menu *Insert\Ftp Action*. The new action is appended to the active project and the property bar shows the properties of the action.
3. Change the name.
4. Indicate the data transfer direction.
5. Indicate the destination and the source in **LocalPath** and **RemotePath**.
6. Indicate the mask for source in **FileMask** property.
7. Verify **User** and **Password** for ftp connection.
8. Indicate actions in **OnOk** and **OnError** property.

For example, we need to transfer a signature to a Server, so we need a **Drawing Control**, two buttons to delete and save items (**Button Controls**), input and output field of files' name (**Edit Control**) that saved the signature, a button to send data (**Button Control**), as well as two **Static Controls** to identify input field.

Create a new project named *Send_picture* and add a screen named *sScreen* where you place the Controls (Fig. 87) :

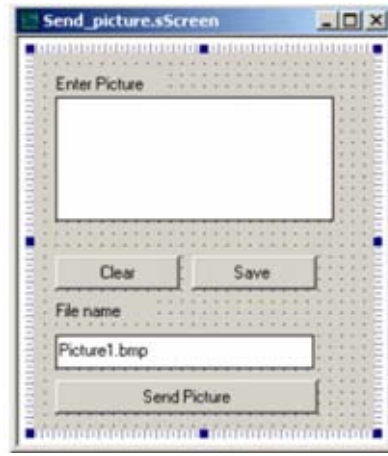


Figure 87

Add to the project a new variable named *vPictureName*. This variable is designed to save the file's name. In its **Data** property, indicate *Picture1.bmp*. In **Data** property for **Edit Control** indicate the same value (*Picture1.bmp*). Indicate the following parameters for **Drawing Control** (Figure 88) :

Fig. 88 : the Drawing Control parameters

Now, add **Operation Action** with the next parameters (Fig. 89) :

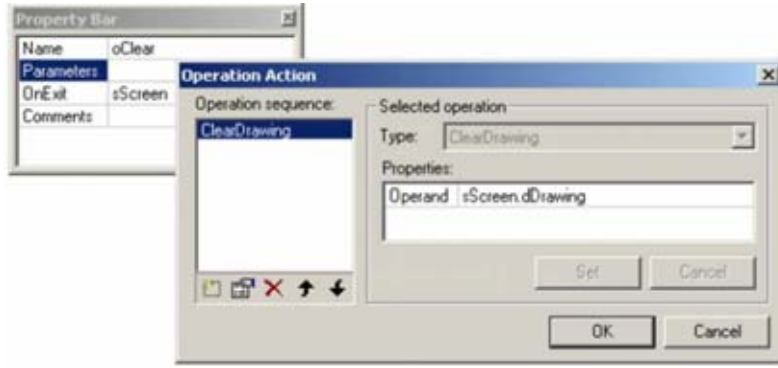


Fig. 89

The purpose of this action is to clear or store in memory the operator signature.
Add a final action to store the **Edit Control** value in the *vPictureName* variable.

Figure 90

When this action is finished, the *oSave* Action comes. Assign to *Save* button *oFileName* Action and to *Clear* button the *oClear* one.
Add to the project a **FTP Action**. In **LocalPath** field, indicate **\\Windows\Start Menu\.** In **RemotePath** property indicate the name or IP address of the Ftp server, for example (Fig. 90).

Fig. 91 : properties of FTP Action

In **User** and **Password** property indicate the name and the password used to connect to Server. If you don't want to let the copied file on PDA, set "Yes" in **DeleteSrc** property. If you don't want to send all files, indicate *.bmp in **FileMask** property. Add to the project the message window that will appears if there is a copy error or if the cancel button is pressed. Select this window in **OnError** property.
Now, we have to assign the ftp Action to **Send Picture** button and indicate the screen's name *sScreen* in **OnStart** field of Project and determine the type of your PDA. The project is finished. Press **Generate**. Start the program on PDA and enter the signature in **Drawing Control** after the screen looks like this (Figure 92) :

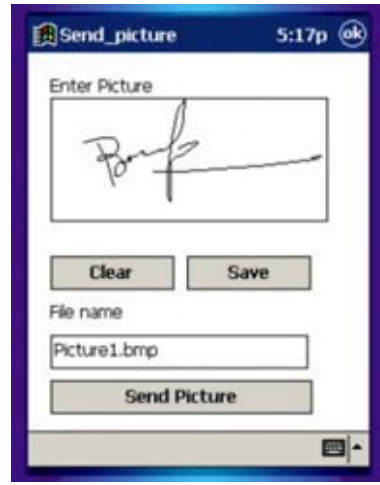


Fig. 92 : the application is launched on PDA

Try the *Clear* button and enter another signature you saved and then press *Send Picture* button. Now, go to the server that you indicated in ftp action and verify if there is Picture1.bmp. file in the main directory. Try to change the file name and send figure. Verify it has been sent on PDA.